MASTER EN SCIENCES INFORMATIQUES

Etude critique des performances de l'éditeur de liens link

Adans, Christian

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

ETUDE CRITIQUE DES PERFORMANCES DE L'EDITEUR DE LIENS LINK.

Christian Adans

Mémoire présenté en vue de
l'obtention du grade de
Licencié et Maître en
Informatique.

Année Académique 1980-1981

AVANT-PROPOS.

Avant-propos.

Je tiens à exprimer toute ma gratitude envers Monsieur J. Ramaekers pour les directives et les conseils qu'il a bien voulu me donner tout au long de mon travail.

Je suis très reconnaissant à la firme Digital Equipment Corporation pour m'avoir permis d'effectuer un stage dans son département Software Services à Bruxelles. Je tiens à remercier tout particulièrement Monsieur R. Haentjens pour son aide constante et ses critiques constructives.

Je remercie Messieurs J. Pierson et M. Debar du Centre de Calcul des Facultés. Leurs conseils furent très précieux.

Je suis particulièrement reconnaissant à Monsieur Ph. De Rivet pour avoir bien voulu discuter avec moi certains aspects des mesures de performances propres à mon problème.

Je remercie Monsieur R. Verhaeghe dont les critiques m'ont été très utiles durant la rédaction de ce mémoire.

Je remercie également Messieurs F. Focant et J. Fripiat pour m'avoir donné accès à quelques-uns de leurs programmes et me permettre ainsi de compléter la charge test pour le LINK.

Je suis très reconnaissant à Monsieur Ban Tri Han du Service d'Informatique de l'Institut Montéfiore pour m'avoir fourni une copie de l'étude de Monsieur Ph. Lemaire sur le LINK.

Enfin, je tiens à remercier Madame Labidi pour le soin et la diligence apporté à la dactylographie de ce mémoire.

TABLE DES MATIERES.

TABLE DES MATIERES.

TABLE DES MATIERES.

INTRODUCTION.

Introduction.

"Les systèmes d'exploitation sont constitués autour d'un ensemble complexe de programmes destinés à optimiser la gestion des ressources de l'ordinateur et à faciliter la tâche du programmeur." ([BCM], p. 2)

Ces deux objectifs ne sont cependant pas toujours compatibles. Un exemple de ce fait est l'éditeur de liens LINK. Il facilite la tâche du programmeur moyennant la consommation de ressources coûteuses telles que temps CPU, place mémoire, etc ... . Il est l'un des cinq programmes les plus utilisés du DECsystem-20 et l'un des plus coûteux en ressources consommées. Il constitue une charge relativement importante pour le système et toute amélioration peut être avantageuse.

Ce mémoire à pour but de proposer l'une ou l'autre possibilité d'améliorer les performances du LINK.

Pour ce faire, nous allons procéder à une étude de ses performances. Précisons que nous avons choisi d'étudier la consommation en temps CPU du LINK. Ceci se justifie par le fait que le département Software Engineering de la firme Digital Equipment Corporation a déjà entrepris d'apporter diverses améliorations au LINK (entre autres, du point de vue de son comportement en mémoire virtuelle). Dès lors, il nous a semblé intéressant de nous diriger vers d'autres possibilités qui, a notre connaissance, n'ont pas encore été prises en considération.

La première partie de ce mémoire est consacrée à une description des techniques d'édition de liens en général et du LINK en particulier. Nous ne manquerons pas de faire référence aux concepts décrits dans cette partie.

Dans une seconde partie, nous développerons les méthodes et les outils nécessaires à la réalisation des mesures de performances. Il est très important de procéder méthodiquement et de justifier les mesures proposées étant doné que celles-ci sont généralement coûteuses en temps et en moyens.

Les résultats de ces mesures nous permettront, dans la troisième partie, de déterminer dans quels sous-ensembles du LINK une amélioration est souhaitable. Nous y discuterons certaines propositions du point de vue de leur efficacité ainsi que de leur possibilité de réalisation.

PREMIERE PARTIE.

PRESENTATION DE L'OBJET DE L'ETUDE:

L'EDITEUR DE LIENS LINK.

1. GENERALITES.

Avant de commencer l'étude de performances proprement dite, nous devons évidemment introduire l'objet de ces mesures. Cependant, nous resterons à un niveau assez global. Dans la suite, chaque fois que cela s'avère nécessaire, nous fournirons des descriptions plus détaillées.

D'une manière assez générale, quelle est la raison d'exister d'un éditeur de liens ?

1.1. Le problème de l'édition de liens.

L'éditeur de liens s'insère dans une chaîne de programmes utilitaires destinés à faciliter le travail du programmeur (cf. figure 1-1). En effet, celui-ci peut écrire son programme en un langage dit "évolué" (ou "symbolique") parfois plus proche d'un langage naturel et que la machine ne peut pas exécuter immédiatement.

Le problème consiste donc à fournir un programme "exécutable" en partant d'un programme "source".

Dans une première phase, le programme source doit être traduit en langage-machine. On dit qu'il doit être "compilé". Le code-source doit être traduit en code-machine et les symboles doivent être résolus (traduits) en adresses.

A ce stade, le programme n'est pourtant pas encore exécutable. Il n'est pas encore complet ! En effet, le travail du programmeur est encore facilité par le fait qu'il peut faire appel à toute une série de "fonctions standards" pré-programmées.

Parmi les fonctions pré-programmées, on peut trouver, notamment, des fonctions mathématiques (telles que SINUS, COSINUS, RACINE CARREE, etc ...) ou d'entrée/sortie. Ces fonctions se trouvent rassemblées dans des "librairies" de programmes accessibles par l'ensemble des utilisateurs. Dans son programme, le programmeur pourra faire référence à ces fonctions à l'aide du nom

(symbole) sous lequel chacune est définie dans la librairie.

Il est, bien entendu, loisible à l'utilisateur de se constituer sa propre librairie de sous-programmes dont il se sert souvent.

D'autre part, par un soucis de modularisation, on peut vouloir compiler séparément diverses parties d'un programme (programme principal et sous-programmes, par exemple). On parle alors de "modules compilés séparément".

Dès lors, on appellera symbole global (ou externe) un symbole défini dans un module mais utilisable dans d'autres. Par contre, un symbole sera dit local s'il ne peut être utilisé que dans le module où il est défini. Seules les références à des symboles locaux peuvent être résolues à la compilation.

C'est à l'éditeur de liens, dans une phase ultérieure, de résoudre ces références à des symboles externes. Ceci se fera grâce aux renseignements que lui fournira l'utilisateur (renseignements tels que noms des fichiers contenant les programmes compilés, noms de librairies, etc ...).

Finalement, il restera à charger le tout physiquement en mémoire afin d'obtenir le produit désiré : un programme exécutable.

Figure 1-1: Chaîne de traitement d'un programme.

1.2. Quelques solutions.

Après avoir brièvement situé la question de l'édition de liens, voyons maintenant quelques manières de la résoudre (cf. [DB], [PW], [JD]).

1.2.1. La technique du "COMPILE-AND-GO".

Cette technique consiste dans le fait que le compilateur génère directement du code exécutable et le place lui-même en mémoire.

Elle présente comme principal avantage la facilité d'implémentation. En effet, à la fin de la compilation, une simple instruction de branchement vers l'adresse de début du programme permet d'en faire démarrer l'exécution.

Elle présente cependant aussi certains désavantages :

- une partie de l'espace mémoire est inutilisable puisqu'occupée par le compilateur,
- la traduction du programme source doit être répétée à chaque exécution,
- cette technique ne facilite pas la modularisation (Il est plus compliqué de traiter ainsi des programmes décomposés en différents modules).

On se rend compte qu'un compilateur "COMPILE-AND-GO" n'est réellement avantageux que pour des programmes relativement petits. Lorsqu'un programme atteint une certaine taille, il faut faire appel à une autre méthode.

1.2.2. L'utilisation de fichiers intermédiaires.

On peut éviter certains désavantages de la technique "COMPILE-AND-GO" en plaçant le résultat de la compilation dans un fichier intermédiaire. Ce type de fichier est appelé "fichier objet".

Pour placer le code généré en mémoire, il faut ensuite faire appel à un "chargeur" ("LOADER").

Ainsi, la compilation ne devra plus être effectuée à chaque exécution.

Cette technique admet aussi que des parties de programme soient compilées séparément.

De plus, si tous les compilateurs et assembleurs produisent des fichiers objets répondant à certaines conventions, il est possible d'écrire des sous-programmes dans des langages différents.

Voyons successivement quelques types de chargeurs.

1.2.3. Le chargeur "absolu".

Dans le code généré par les compilateurs, toutes les références sont sous forme d'adresses absolues.

Un tel chargeur est relativement petit et on perd donc moins de place en mémoire.

Un désavantage majeur de cette technique réside dans le fait que les références à des "objets externes" se font grâce à des adresses absolues ! L'utilisateur devra donc connaître l'adresse à partir de laquelle sera chargée, par exemple, une sous-routine. Lorsqu'il fera référence à cette sous-routine, il utilisera cette adresse.

Par conséquent, une modification, même mineure, dans un module risque d'avoir des répercussions sur les autres modules.

1.2.4. Le chargeur "relogeur".

Il s'agit d'améliorer la technique précédente. Pour ce faire, le compilateur doit inclure, dans le module objet, des informations au sujet des sous-programmes externes référencés.

Ceci peut être réalisé en utilisant un "vecteur de transfert". Le principe est le suivant :

Chaque fois qu'un programme fait référence à un sous-programme externe, le compilateur génère une instruction de branchement vers un élément du vecteur de transfert. Ensuite, le chargeur remplacera, dans ce vecteur, le nom du sous-programme par une instruction de branchement vers l'adresse où le code correspondant sera placé.

D'autre part, le compilateur fournira des "informations de relogement".

En effet, le code généré est dit "relogeable" i.e. toutes les adresses sont relatives à l'adresse 0 (qu'il s'agisse d'adresses de données ou de branchement). Ces adresses dépendent de la zone mémoire où le programme sera chargé.

L'ajustement de ces adresses s'appelle le "relogement". Il consiste à ajouter la constante de chargement (i.e. l'adresse à partir de laquelle le programme est chargé) aux adresses des instructions qui le nécessitent (une valeur immédiate ne nécessitant pas de relogement).

Les "informations de relogement" indiquent si oui ou non il y a lieu d'ajuster une adresse.

Un avantage évident de cette technique est que le programmeur peut écrire ses (sous-)programmes sans se soucier de l'endroit où ils seront chargés en mémoire.

De plus, une modification dans un module ne nécessite pas de remaniement des autres.

Par contre, si l'emploi d'un vecteur de transfert permet les branchements vers des sous-programmes externes, il reste la question de l'accès à des données externes. En effet, cette technique ne permet pas l'accès à des segments de données partagés entre différents modules.

1.2.5. Le relieur.

Dans ce cas, un compilateur fournit, avec le module objet, des informations de relogement et des renseignements relatifs aux symboles qu'il a rencontrés mais dont il n'a pas pu trouver la définition. Pour chacun de ces symboles, il donne la liste des instructions qui y font référence. Il fournit également la liste des symboles définis dans le module mais accessibles par d'autres.

C'est le rôle d'un relieur (ou éditeur de liens) de résoudre les références aux symboles indéfinis et d'assurer l'ajustement des adresses. Il fournit un programme "chargeable" placé dans un fichier intermédiaire.

Ensuite, un simple chargeur absolu suffit pour charger le tout en mémoire centrale.

Il arrive que relieur et chargeur soient rassemblés en un seul utilitaire. On est alors en présence d'un relieur-chargeur ("LINKING-LOADER").

2. PRESENTATION DU PROGRAMME LINK.

Nous avons présenté la question et décrit brièvement quelques solutions (Il en existe d'autres). Poursuivons maintenant en précisant comment elle a été résolue dans le cadre du DECsystem-20.

Le LINK est du type relieur-chargeur. Sa fonction est de rassembler des modules compilés séparément pour en faire un programme exécutable. Pour ce faire, il doit répondre aux objectifs suivants (cf. [JD], p. 149):

- Allouer l'espace mémoire nécessaire au code et aux données (Allocation),
- Résoudre les références symboliques entre modules objets (Edition de liens),
- Assurer l'ajustement des parties adresse des instructions (Relogement),
- Placer physiquement le code et les données en mémoire (Chargement).

Nous décrirons successivement ce que l'utilisateur fournit au LINK (l'"INPUT"), ce que le LINK produit (l'"OUTPUT") pour en arriver, finalement, à la structure globale du programme.

2.1. Les données d'entrée du LINK.

En entrée, le LINK accepte (cf. [LRM], [PL]):

- les modules objets produits par les compilateurs,
- les librairies de modules pré-compilés,
- les commandes de l'utilisateur.

2.1.1. Le module objet.

Les compilateurs du DECsystem-20 produisent des modules objets qui sont tous formatés selon le même principe.

Ces modules objets contiennent, principalement :

- Le code généré par le compilateur ou l'assembleur,
- Des informations nécessaires au placement de ce code en mémoire,
- La liste des symboles globaux et des instructions qui y font références.

La structure des modules objets est la suivante :

Ils sont formatés en un certain nombre de blocs appelés "REL BLOCKS". Ceux-ci sont eux-mêmes formés d'un certain nombre de mots de 36 bits.

Il existe des REL BLOCKS de types différents, selon leur contenu, mais leur structure fondamentale est toujours la même (cf. figure 2-1).

Le premier mot de chaque bloc est appelé "HEADER WORD". Il contient le numéro du type du bloc dans sa moitié gauche, et la longueur du bloc (en nombre de mots) dans sa moitié droite.

Le second mot est un "RELOCATION WORD". Ce mot contient les informations de relogement relatives aux 18 mots suivants.

Tout mot qui n'est ni HEADER WORD ni RELOCATION WORD, est appelé "DATA WORD".

Les deux premiers bits d'un RELOCATION WORD indiquent la "relogeabilité" des deux moitiés du premier DATA WORD. Les troisième et quatrième bits indiquent la "relogeabilité" du deuxième DATA WORD, et ainsi de suite.[1]

---

[1]Un bit à 1 signale que la moitié correspondante du DATA WORD doit être ajustée lors du chargement. Un bit à 0 signale qu'il n'y a pas lieu d'ajuster la moitié correspondante.

Si un REL BLOCK contient plus de 18 DATA WORDS, il y aura un RELOCATION WORD pour chaque groupe de 18 DATA WORDS.

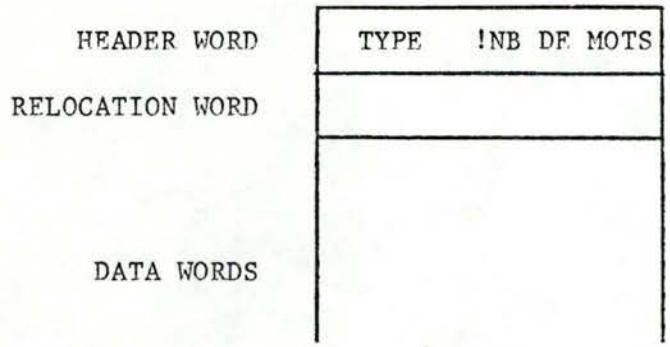


Figure 2-1: Schéma général d'un REL BLOCK.

La nature du contenu d'un REL BLOCK est parfaitement déterminée par le type du bloc. Le tableau de la figure 2-2 donne quelques exemples de types de REL BLOCKS.

Type de bloc	contenu
1	code et données
2	symboles
4	points d'entrée
5	fin
6	nom du programme
7	adresse de départ
....	.....

Figure 2-2: Exemples de types de REL BLOCKS.

Il serait vain de vouloir décrire ici chaque type de bloc en détail. (quelques exemples peuvent être trouvés dans l'annexe 1. Pour plus de détails, nous renvoyons le lecteur au LINK Reference Manual, [LRM], ou à l'étude de Ph. Lemaire, [PL].)

2.1.2. Les librairies.

Une librairie est un fichier contenant un ou plusieurs modules objets avec leurs points d'entrée (cf. [LRM]).

Précisons que l'on peut différencier entre les librairies-systèmes, d'une part, et les librairies propres à un utilisateur, d'autre part.

A chaque compilateur est associé une librairie-système. Un compilateur génère des appels à certaines routines (d'Entrée/Sortie p.ex.) contenues dans la librairie correspondante.

L'éditeur de liens se charge de rassembler le tout en un programme exécutable, en mémoire. Pour ce faire, le LINK parcourt séquentiellement la (les) librairie(s) correspondante(s) au(x) langage(s) rencontré(s) et ne charge un module que si cela lui permet de résoudre une référence globale.

Les librairies-systèmes sont "consultées" automatiquement à la fin du chargement (à condition, bien sûr, qu'il reste au moins une référence non résolue).

Dans le cas des librairies d'utilisateurs, il y a lieu de spécifier explicitement le moment de la "consultation".

Dans le cas le plus simple, une librairie n'est rien d'autre qu'une série de modules relogeables mis bout à bout. Dans cette éventualité, le LINK doit la parcourir toute entière.

Il y a cependant moyen d'accélérer la recherche dans une librairie en lui adjoignant un ou plusieurs blocs d'index (REL BLOCK de type 14). On parlera donc de librairies indexées ou non indexées[2].

De toute façon, une librairie est parcourue sans retour en arrière.

---

[2]Nous reviendrons sur ce mécanisme dans la Troisième Partie.

2.1.3. Les commandes.

L'utilisateur contrôle l'édition de liens-chargement par des commandes (cf. [LRM]).

Une ligne de commandes est constituée de spécifications de fichiers et/ou d'options ("SWITCHES").

Les "SWITCHES" permettent :

- soit de préciser ce qu'il y a lieu de faire avec les fichiers cités,
- soit d'obtenir des informations sur le déroulement des opérations.

Lorsque le LINK lit la fin d'une ligne de commandes (touche "carriage return"), il traite complètement cette ligne avant d'en accepter une nouvelle.

Il s'établit donc une sorte de dialogue entre l'utilisateur et le LINK. Ce dialogue se poursuit jusqu'à ce que l'utilisateur introduit le "SWITCH" "/GO". Dans ce cas, le LINK réalise la recherche dans la(les) librairies(s)-système(s) et termine le chargement (cf. figure 2-3).

```
§LINK
*PROGRM.REL                    !l'utilisateur demande au LINK de
                               !traiter le fichier objet PROGRAM.REL
*SUBRT.REL                     !l'utilisateur demande le char-
                               !gement du fichier SUBROUT.REL
*LIBRAR.REL/SEARCH             !la librairie de l'utilisateur
                               !doit être parcourue
*/UNDEFINED                    !l'utilisateur demande que lui soit
                               !fournie la liste des symboles non
                               !encore définis
[LNKUGS  9 undefined global symbols    !le LINK répond qu'il
         STOP.                         !y a 9 symboles indé-
         RESET.                        !finis
         IN.
         FIN.
         EXIT.
         IOLST.
         OUT.
         ADJ1.
         DSQRT.]
*/ERRORLEVEL:0                 !l'utilisateur demande l'affichage de
                               !tous les messages à l'ecran.
[LNKSNL   Scanning new command line]   !le LINK répond qu'il
                                       !attend une nouvelle ligne
                                       !de commandes
*/GO                           !l'utilisateur est satisfait et
                               ! donne l'ordre de terminer
[LNKLDS   LOAD segment]                !le LINK indique le dérou-
[LNKLMN   Loading module FORINI]       !lement des opérations en
[LNKLMN   Loading module FORPSE]       !fournissant la liste des
[LNKLMN   Loading module DSORT.]       !modules  recherchés
[LNKLMN   Loading module ADJ1.]        !en librairie-sytème
[LNKLMN   Loading mudule CFRXIT]
[LNKEXS   EXIT segment]                !le LINK entre dans son
[LNKSST   Sorting symbol table]        !segment LNKXIT
[LNKSTC   Symbol table completed]
[LNKFIN   LINK finished]               !le programme exécutable se
                                       !trouve en mémoire, prêt
EXIT                                   !pour l'exécution
§
```

Figure 2-3: Exemple de dialogue entre le LINK et l'utilisateur.

2.2. Les résultats du LINK.

2.2.1. Le résultat primaire.

Principalement, le LINK fournit le programme exécutable encore appelé "CORE IMAGE". Dans le programme exécutable, toutes les références symboliques ont été résolues en des adresses absolues en mémoire.

A la fin de l'édition de liens-chargement, le programme exécutable se trouve dans l'espace d'adressage de l'utilisateur (en lieu et place du LINK). Il peut être exécuté et/ou sauvé sur fichier en vue d'exécutions futures. Cette dernière possibilité permet d'aller plus vite car on épargne l'édition de liens. Le désavantage de ce gain de temps est que l'on consomme plus de place sur mémoire secondaire. En effet, les fichiers exécutables prennent généralement beaucoup de place.

2.2.2. Les résultats secondaires.

Lors de l'édition de liens-chargement, l'utilisateur peut demander la création de divers fichiers.

Il en est ainsi du fichier "MAP". Il s'agit d'une "carte" indiquant l'emplacement et la taille des différents modules en mémoire. A titre facultatif, on peut également y trouver la liste, triée par ordre alphabétique, des symboles définis dans chacun des modules, ainsi que leurs valeurs.

Finalement, il existe encore la possibilité d'obtenir un fichier "LOG" contenant les messages relatifs au travail du LINK (cf. [LRM]).

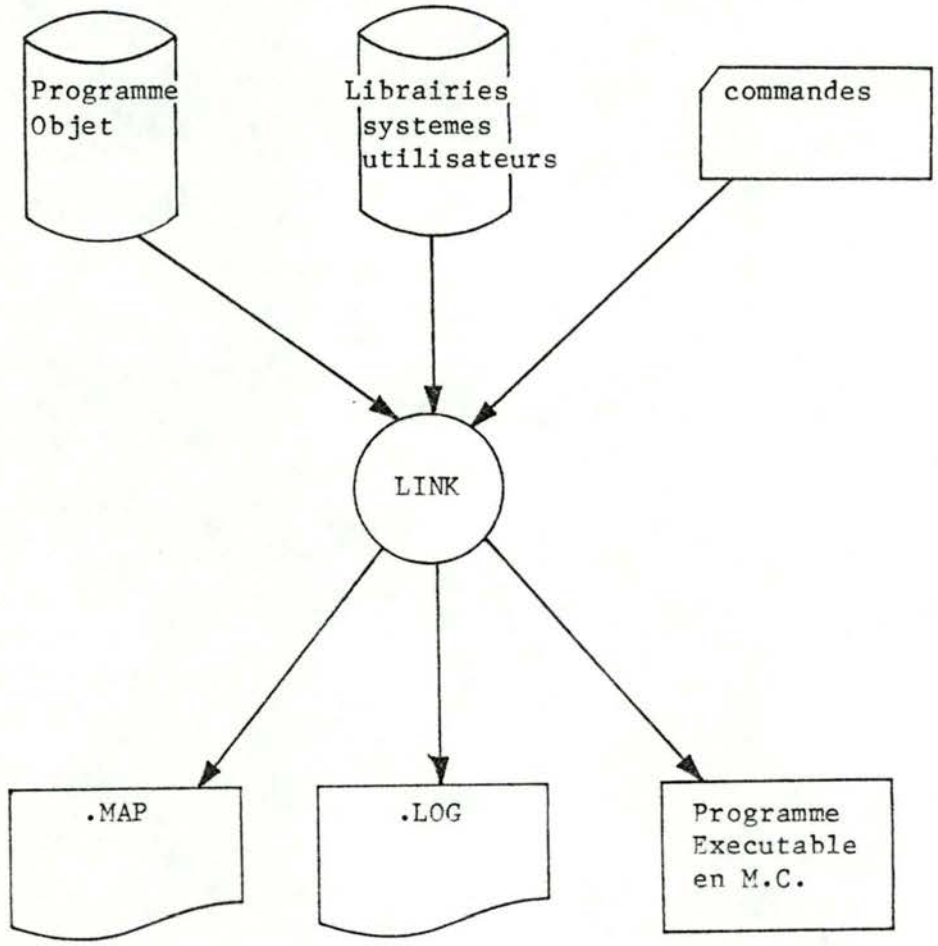


Figure 2-4: Input et Output du LINK.

2.3. La structure générale du LINK.

2.3.1. Découpe en modules fonctionnels.

Le LINK est un programme composé d'environ 45.000 instructions assembleur. Il est décomposé en 20 modules. Chaque module correspond plus ou moins à une fonction à remplir par le LINK.

Décrivons brièvement les modules les plus importants (cf. [PL] pour des descriptions plus détaillées).

LNKINI[3] Ce module réalise toutes les initialisations nécessaires (initialisations des zones de travail, des tables de symboles, etc ...).

LNKSCN Ce module est un interface entre le LINK proprement-dit et le module .SCAN. .SCAN est un ensemble de routines assurant la lecture des commandes introduites au terminal. (La plupart des compilateurs ainsi que divers autres utilitaires des DECsystem-10 et 20 utilisent .SCAN pour la lecture des commandes) LNKSCN transforme une ligne de commande en une chaîne de blocs descripteurs de fichiers et de "SWITCHES".

LNKLOD Réalise le chargement des fichiers spécifiés en appellant les modules de traitement nécessaires.

LNKLWD traite les "SWITCHES".

LNKOLD assure le traitement des "REL BLOCKS" selon leur type.

LNKXIT assure la terminaison du chargement i.e. écrit un fichier exécutable s'il y a lieu de le faire et, éventuellement, lance l'exécution du programme. De toute façon, il réalise le "suicide" du LINK c'est-à-dire la remise à zéro de l'espace préalablement occupé par le LINK.

Il existe d'autres modules, d'importance moindre, qui ne sont pas décrit ici.

---

[3]Tous les noms de modules sont préfixés par "LNK" et suffixés par une abréviation générique en 3 lettres et/ou chiffres.

2.3.2. L'enchaînement des modules.

Cet enchaînement peut être décrit sous la forme d'un "pseudo-algorithme" ainsi que sous la forme d'un organigramme (cf. figures 2-5 et 2-6).

```
INITIALISATION;
LIRE UNE LIGNE DE COMMANDES;
TANT QUE "/GO" N'A PAS ETE RENCONTRE
        TRAITER SWITCHES
        CHARGER FICHIERS
        LIRE UNE LIGNE DE COMMANDES;
RECHERCHER SYMBOLES NON DEFINIS EN LIBRAIRIES-SYSTEME;
FINALISATION;
```

Figure 2-5: Pseudo-algorithme du LINK.

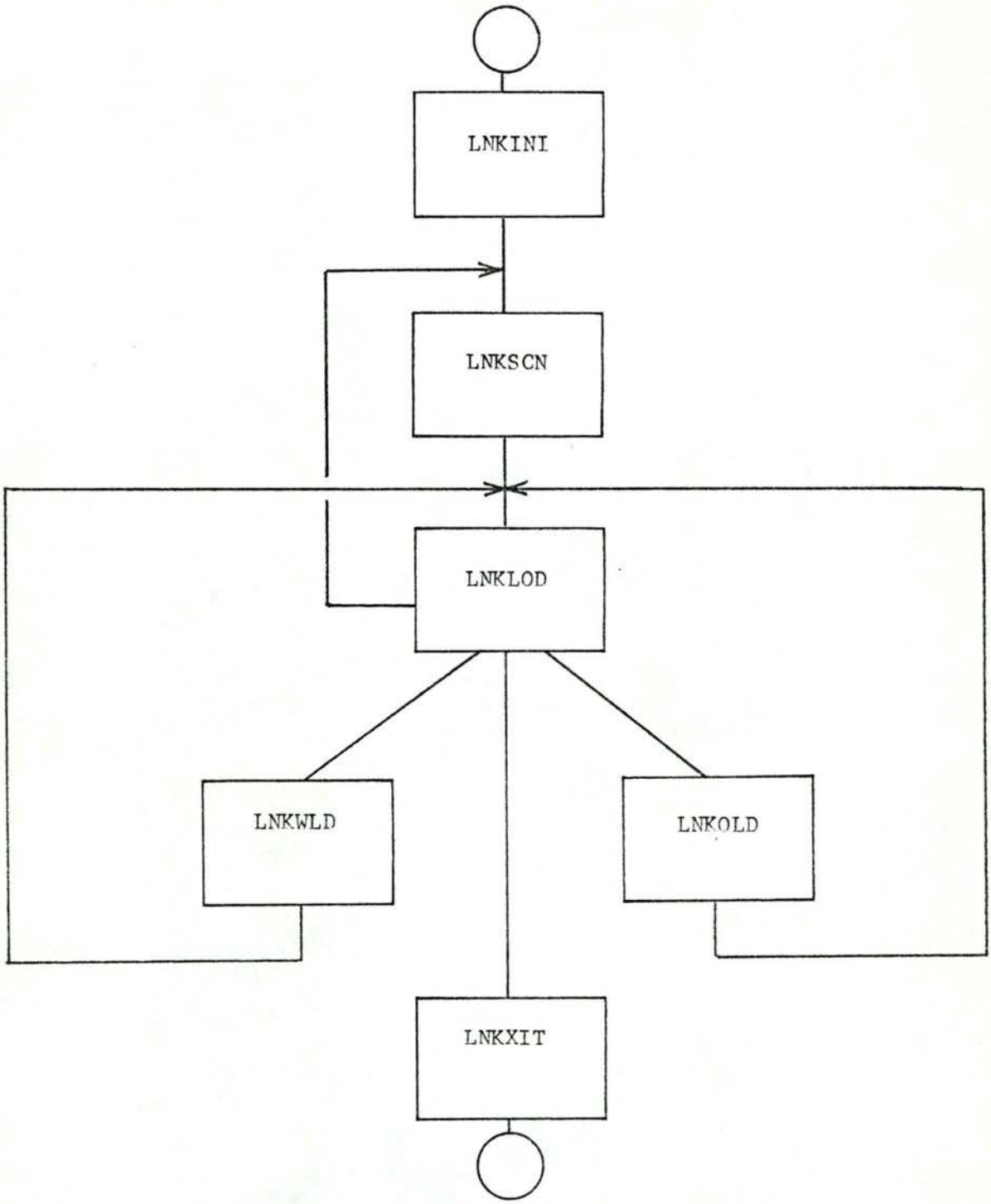


Figure 2-6: Organigramme général du LINK.

2.4. L'origine du LINK.

Il importe de remarquer ici que le LINK a été écrit, à l'origine, pour le DECsystem-10. Cet ordinateur fonctionne sous le système d'exploitation TOPS-10.

Notre analyse du LINK a été réalisée sur le DECsystem-20 du Centre de Calcul des Facultés de Namur. Celui-ci fonctionne sous le système d'exploitation TOPS-20.

La firme Digital Equipment Corporation désire garder, dans la mesure du possible, les mêmes utilitaires sur les deux systèmes, DECsystem-10 et DECsystem-20. Or, les appels-systèmes TOPS-10 ("UUO" ou "UNIMPLEMENTED USER OPERATION") diffèrent fortement des appels-systèmes TOPS-20 ("JSYS" ou "JUMP-TO-SYSTEM").

Le problème a été résolu par l'utilisation d'un interface: PA1050 encore appelé "COMPATIBILITY PACKAGE".

Imaginons un programme s'executant sous TOPS-20. Lors de la première tentative d'exécution d'un appel-système TOPS-10, le système d'exploitation TOPS-20 charge, dans l'espace d'adressage du programme, l'interprèteur PA1050. Celui-ci interprète les appels-systèmes TOPS-10 en appels-systèmes TOPS-20. Dans la suite, toute nouvelle tentative d'exécution d'un "UUO" résulte en un déroutement direct vers PA1050.

PA1050 est toujours placé dans les pages 700b[4] à 730b de l'espace d'adressage. Par conséquent, un programme faisant appel à PA1050 ne peut pas utiliser ces pages.

---

[4] A partir d'ici, un "b" suivant un nombre signale qu'il s'agit d'un nombre octal.

DEUXIEME PARTIE.

LES MESURES DE PERFORMANCES.

3. MESURES DE PERFORMANCES - GENERALITES.

3.1. Les buts.

Les mesures de performances se font en fonction d'un but bien déterminé. Le nôtre est de trouver des possibilités d'améliorer les performances d'un logiciel en diminuant sa consommation en temps CPU.

Le choix des mesures à effectuer peut être très vaste. Il s'agit donc d'être prudent car les mesures sont généralement coûteuses et longues. Bien que la question du financement ne se pose pas de manière cruciale dans le cadre de ce mémoire, le facteur temps, lui, intervient de façon sensible. Des mesures "non réfléchies" peuvent introduire une perte de temps considérable.

L'important est donc de bien fixer les buts initiaux. Ensuite, il faudra en déduire un plan des mesures à effectuer.

3.2. Le plan de mesures.

Le plan de mesures doit préciser quelles sont les variables à mesurer et pourquoi. Il s'agit de justifier, à l'avance, les mesures en précisant ce que l'on pourra faire avec les résultats.

Pourtant, et l'expérience le montre, certains résultats seront totalement imprévisibles. Par conséquent, l'explication a posteriori s'avèrera souvent inévitable (cf. [DF]).

Ceci va introduire une certaine récurrence dans les mesures, en ce sens que les résultats d'une phase de mesures nécessiteront une nouvelle phase pour pouvoir être convenablement expliqués.

3.3. La démarche de la mesure.

Conceptuellement, le procédé peut être décrit comme suit (cf. [PHD]).

Imaginons un "système" acceptant, en entrée, ce que nous appellerons une "charge", sans préciser plus. Ce système fournit, en sortie, une certaine "résultante", moyennant la consommation de certaines ressources.

En toute généralité, supposons que l'on veuille mesurer ce système dans le but d'en connaître ses performances.

Dès le départ, le mesureur a une certaine perception du fonctionnement du système. Il modélise ce système en ce sens qu'il abstrait du réel ce qui lui semble intéressant. Volontairement, le mesureur simplifie le système réel afin de pouvoir mieux l'appréhender.

La modélisation est une étape intellectuelle qui se traduit, pratiquement, par le choix des variables à mesurer (variables d'entrée et variables de sortie).

En principe, le modèle doit fournir des valeurs de variables de sortie en fonction des valeurs des variables d'entrée.

Les mesures vont pouvoir valider ou invalider le modèle utilisé. C'est ici que se situe l'origine de la récurrence dont il a été question plus haut. En effet, c'est la confrontation des valeurs des variables de sortie mesurées avec celles fournies par le modèle qui est à la base du "feed back".

Grâce à cela, on peut finalement arriver à un modèle acceptable de la réalité.

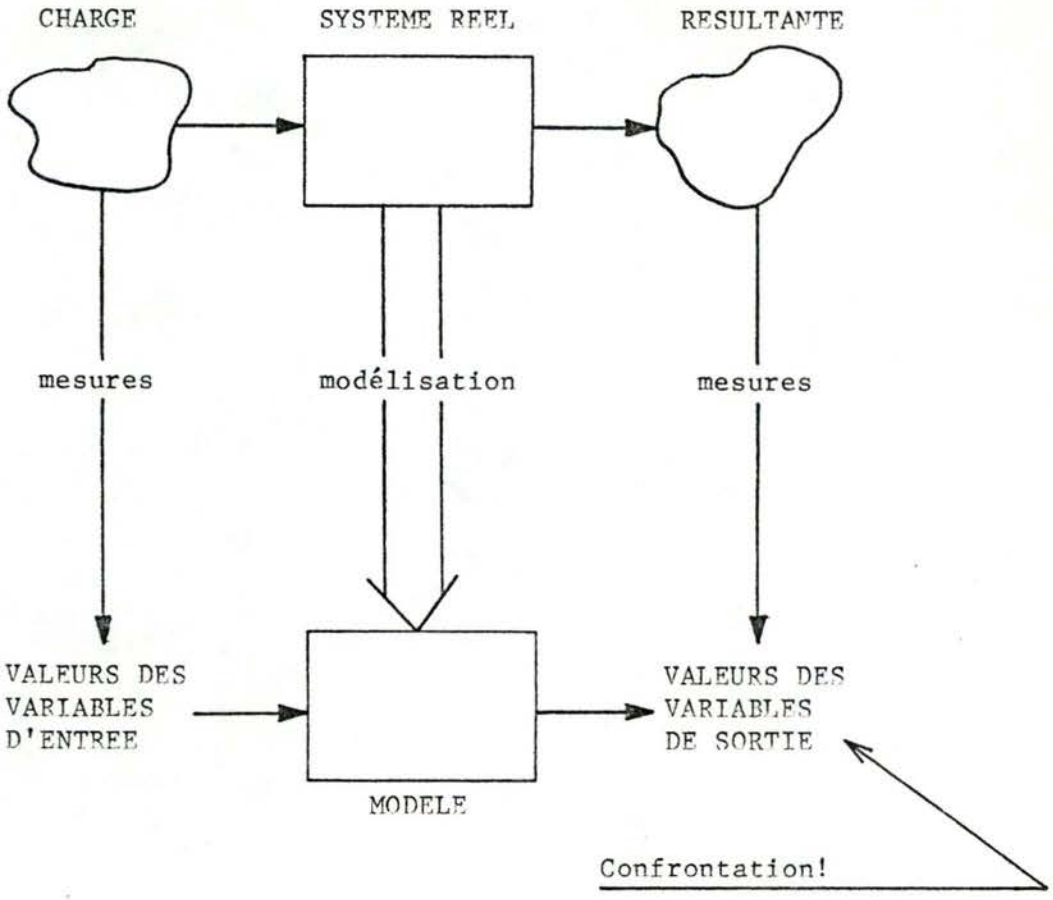


Figure 3-1: La démarche conceptuelle de la mesure.

Il faut cependant insister sur le fait que notre but n'est pas de constuire un tel modèle du LINK ! C'est la démarche de la mesure qui nous intéresse. Dès lors, un modèle peut être assez vague, du moment qu'il permet de déduire les variables à mesurer. Si, dans la suite, les résultats ne correspondent pas à l'idée que l'on avait au départ, il y a lieu de la modifier pour pouvoir expliquer les anomalies.

4. ETABLISSEMENT D'UN PLAN DE MESURES.

4.1. Que mesurer - Pourquoi.

La démarche décrite ci-dessus nous permettra de prévoir des modification éventuelles au plan de mesures que nous développerons dans ce chapitre.

Il s'agit de dégager les grandeurs intéressantes à mesurer. Les méthodes et outils nécessaires feront l'objet des chapitres suivants.

Le but que nous nous sommes fixés est de rechercher des possibilités d'améliorer le LINK au point de vue de sa consommation en temps CPU.

Pour ce faire, il est utile de connaître, d'abord, les performances actuelles du LINK. Il s'agit donc de mesurer une variable résultante : le temps CPU consommé pour l'édition de liens et le chargement des programmes objets. Ce sont ces programmes qui constituent la charge du LINK.

Dans une première étape, nous allons nous intéresser au LINK vu comme une "boîte noire" c'est-à-dire décrire son comportement par rapport à la charge sans tenir compte de la structure interne du programme.

Dans une second e étape, nous analyserons ce comportement plus en profondeur, en tenant compte de cette structure. Par opposition à la première étape, nous dirons que le LINK est vu comme une "boîte blanche".

4.2. Le LINK vu comme une "boîte noire".

Le LINK est ici considéré comme une "boîte noire" à laquelle un utilisateur fournit des programmes compilés en entrée et qui en construit un programme exécutable moyennant la consommation de certaines ressources (dont la ressource temps CPU).

L'idée initiale (ou le modèle initial) est que certains paramètres de la charge (ou variables d'entrée) influencent de manière significative la consommation en temps CPU lors de l'édition de liens-chargement.

Dès lors, il semble intéressant de caractériser la charge du LINK à l'aide de ces paramètres.

A priori, la taille et le nombre de symboles globaux des programmes sont susceptibles d'influencer la consommation en temps CPU.

En effet, les fonctions du LINK sont :

- de résoudre les références externes (édition de liens),
- d'ajuster les parties adresses des instructions et de placer, physiquement, le code et les données en mémoire (relogement et chargement).

Il nous paraît donc que ces paramètres peuvent constituer une mesure du travail à effectuer par le LINK.

Dans une première étape, la mesure de ces paramètres nous permettra de caractériser (décrire) la charge de travail du LINK. Ceci nous sera utile afin de composer un échantillon de charge pour la réalisation de mesures plus approfondies.

Dans une seconde étape, une description des performances actuelles du LINK permettra de vérifier si une modification éventuelle est réellement avantageuse.

Le terme "taille d'un programme" nécessite cependant encore quelques précisions.

Le nombre de mots dans les REL BLOCKS de type 1 (code et données) des modules objets peut, éventuellement, servir de taille. Il faudrait connaître ce nombre pour tous les modules objets, y compris ceux des librairies.

Une seconde possibilité consiste à mesurer la taille du programme exécutable ("CORE IMAGE SIZE"), et ce malgré que l'on puisse la classer du côté des variables résultantes plutôt que du côté des variables d'entrée.

Le résultat devant être sensiblement le même pour ces deux méthodes, c'est, finalement, la facilité d'implémentation qui permettra de trancher.

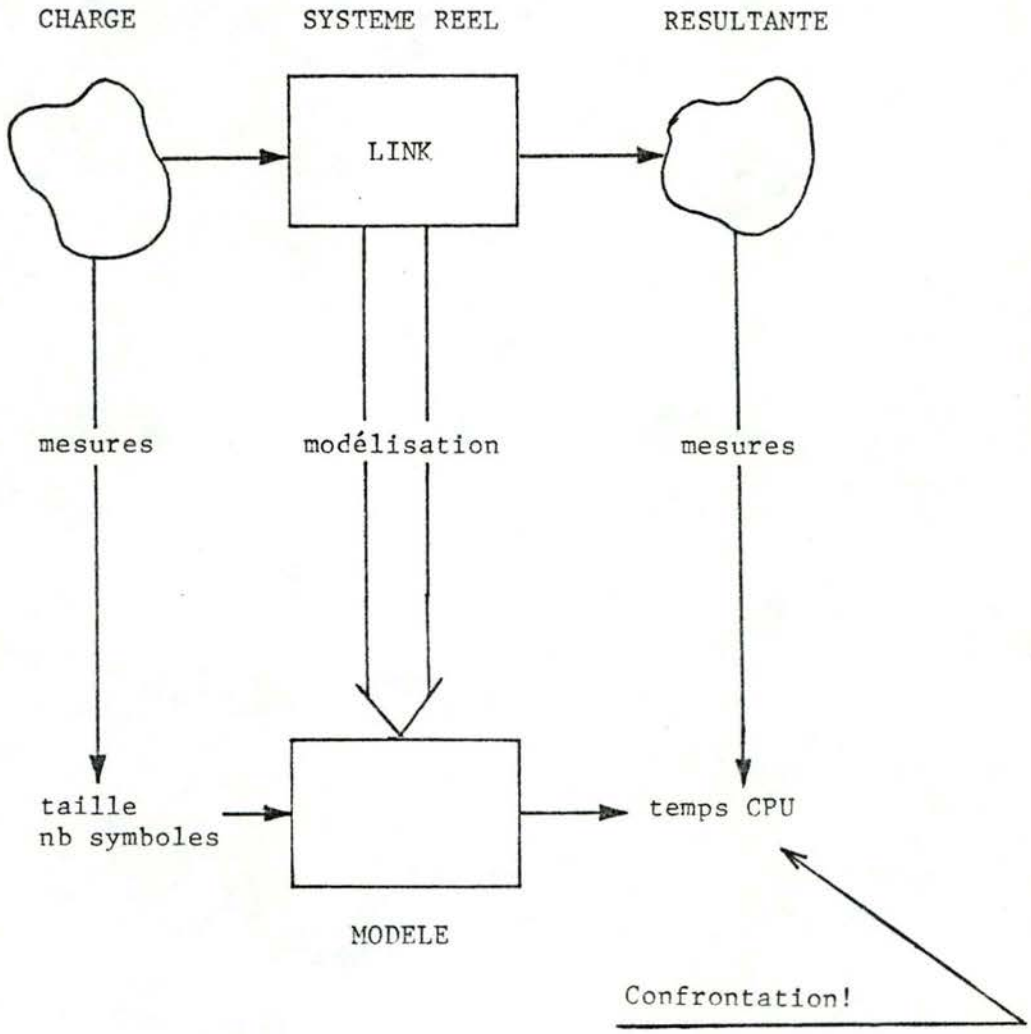


Figure 4-1: L'application de la démarche au cas du LINK.

4.3. Le LINK comme "boîte blanche".

L'idée sous-jacente à cette phase est que, dans la plupart des programmes, plus de 50% du temps CPU sont imputables à moins de 4% du code (cf. [DK]).

Dès lors, c'est dans ces "endroits coûteux" qu'une amélioration éventuelle s'avèrerait la plus avantageuse.

Nous savons déjà combien de temps CPU le LINK consomme pour une charge donnée. Il s'agit, maintenant, de mesurer comment ce temps est distribué sur les différentes parties du LINK.

Le terme "partie" ne pourra être précisé que lorsque nous décrirons la méthode utilisée.

5. MESURES SUR LE LINK VU COMME UNE "BOITE NOIRE".

5.1. La méthode.

A ce stade, nous savons quelles sont les variables à mesurer et pourquoi.

Nous devons, maintenant, développer une méthode et un outil pour réaliser ces mesures.

Bien que le LINK soit considéré, conceptuellement, comme une "boîte noire", nous sommes obligés de chercher les valeurs désirées (taille et nombre de symboles globaux des programmes et temps CPU consommé) là où elles sont disponibles c'est-à-dire dans le LINK même.

Lors de l'exécution du LINK, une ou plusieurs routines (sondes) devront collecter les informations et les placer dans un "endroit sûr" en vue d'un traitement ultérieur.

La méthode est donc très classique et correspond au schéma de la figure 5-1 (cf. [PHD]).

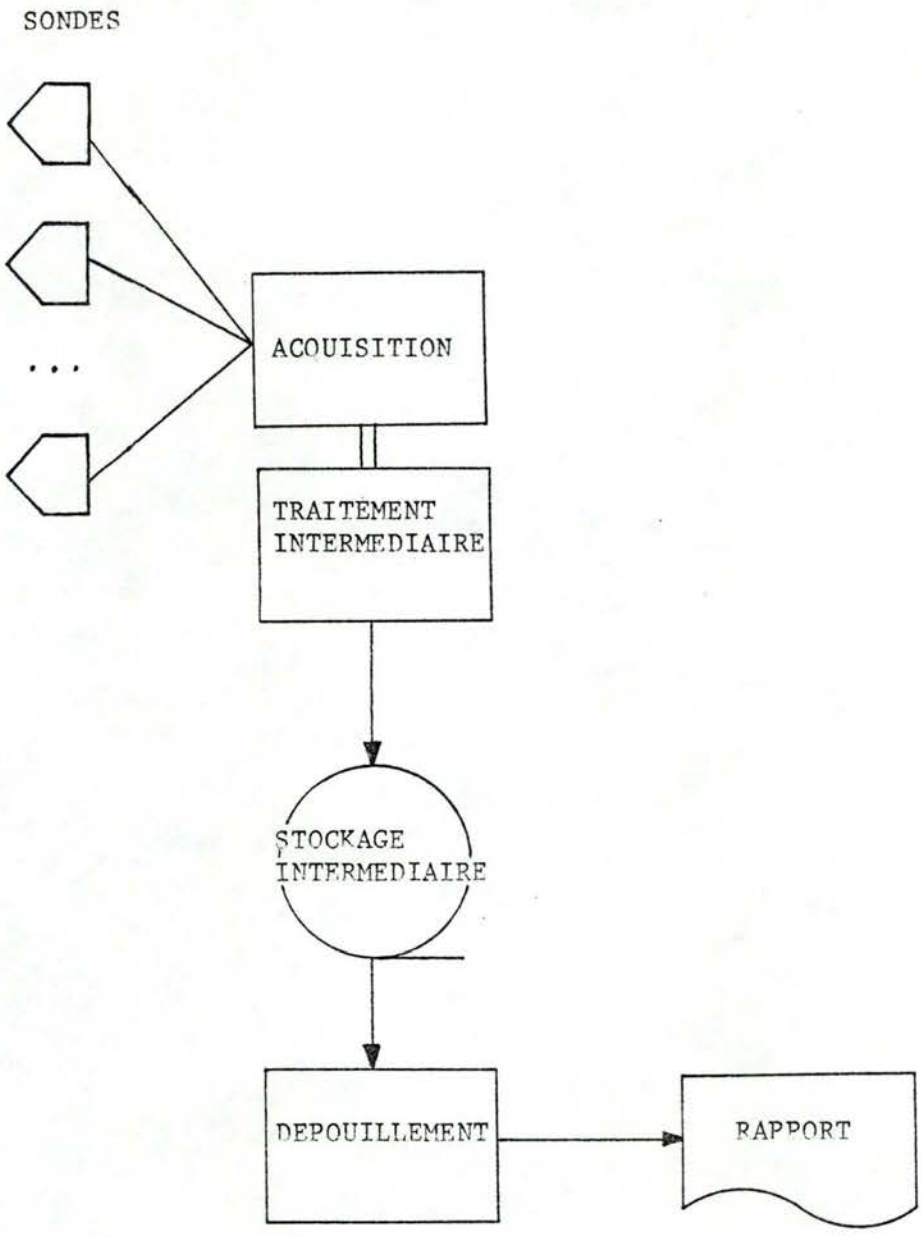


Figure 5-1: Schéma de principe de la méthode.

5.2. L'outil.

La mise au point de l'outil doit répondre à deux questions majeures :

- Comment accéder aux informations ?
- Comment sauver les informations ?

5.2.1. Comment accéder aux informations.

Il apparaît qu'un des modules formant le LINK contient uniquement des données. Il s'agit du module LNKLOW ("LOW SEGMENT DATA BASE"). C'est dans ce module que sont définies toutes les variables utilisées.

Ainsi, on peut y trouver la variable GSYM qui est définie comme contenant, à tout moment, le nombre de symboles globaux rencontrés. Il suffit donc d'échantillonner la valeur de cette variable à la fin de l'édition de liens-chargement.

Par contre, il n'existe pas de variable contenant la taille du programme lié et chargé, et ce quelle que soit la définition de "taille de programme".

La première solution qui consiste à utiliser le nombre de REL BLOCKS de type 1 (code et données) nécessite l'introduction d'un compteur et d'instructions pour incrémenter ce compteur au "bon moment".

D'autre part, la taille du programme exécutable est calculable à partir de certaines variables disponibles dans LNKLOW. Dés lors, c'est cette dernière que nous allons utiliser.

Enfin, le temps CPU peut être mesuré par un emploi judicieux de l'appel-système RUNTM (cf. [MC20], p. 3-199).

Les routines insérées dans le LINK, le calcul de la taille d'un programme

ainsi que les moments des appels aux différentes routines sont décrits dans l'annexe 2.

5.2.2. Comment sauver les informations.

Les routines en question construisent un enregistrement contenant les informations désirées.

Chaque fois que le LINK est utilisé, un tel enregistrement est ajouté à un fichier. Pour la sécurité des informations, il importe d'éviter les conflits d'accès simultanés à ce fichier (voir annexe 2.2.3. pour une description détaillée de cet enregistrement).

5.3. Les résultats.

5.3.1. Présentation des résultats.

Plusieurs périodes de mesures se sont étalées entre décembre 1980 et avril 1981. Nous sommes ainsi en possession de plus de 20.000 observations d'exécutions du LINK.

D'une part, des histogrammes (et les distributions cumulées correspondantes) permettent de décrire la composition de la charge de travail du LINK.

D'autre part, certains graphiques mettent en relation les variables observées.

Tous les histogrammes et graphiques se trouvent rassemblés dans l'annexe 3.

5.3.2. Un problème d'interprétation.

Les graphiques tailles/temps d'une part, et nombres de symboles globaux/temps d'autre part, présentent tous deux un aspect curieux ! En effet, on peut y observer plusieurs "branches".

Ces "branches" indiquent qu'il y a de très fortes variations de consommation en temps CPU pour l'édition de liens-chargement de programmes de tailles semblables. Un phénomène analogue se remarque sur le graphique nombres de symboles globaux/temps.

Avant de pouvoir aller plus loin, nous devons expliquer ce phénomène. Notre idée initiale s'en trouvera légèrement modifiée: La taille et le nombre de symboles globaux des programmes ne sont pas seuls à influencer la consommation en temps CPU du LINK. Nous allons poser et discuter deux hypothèses assez

plausibles:

- Ou bien il s'agit des mêmes programmes dans les différentes "branches" des graphiqes. Dans ce cas, un facteur externe, tel que la charge du système, influence la consommation en temps CPU lors de l'édition de liens.

- Ou bien il s'agit de programmes différents. Dans ce cas, des facteurs internes à ces programmes (en plus de la taille et du nombre de symboles globaux) influencent la consommation en temps CPU.

5.3.3. L'influence de la charge du système.

Une augmentation de la charge du système va de pair avec une augmentation du nombre de défauts de page. Or, le temps passé dans le gestionnaire des défauts de page est compté dans le temps CPU consommé par le processus qui a causé ces défauts de page.

Par conséquent, il est pratiquement indiscutable que la charge du système peut introduire une certaine variation dans la consommation en temps CPU. Pourtant, cette variation est-elle assez importante pour être à l'origine de ce phénomène "de fourche" dans les graphiques ?

Le tableau de la figure 5-2 donne les temps CPU consommés pour les éditions de liens-chargements de divers programmes sous des charges systèmes différentes.[1] Ce tableau montre que la charge du système n'est pas responsable de la présence des différentes "branches". Tout au plus, elle est la cause d'un certain "élargissement" de chaque branche.

Il faut donc trouver une autre explication.

---

[1]La charge-système est mesurée ici, grâce au "ONE MINUTE LOAD AVERAGE". Cet indice de charge est défini comme étant le nombre moyen de processus prêts par minute. Une charge faible correspond, ici, à un indice de 2 à 5 et une charge forte correspond à un indice de 8 à 11.

PROGRAMME NUMERO	TAILLE (en pages)	NB DE SYMBOLES GLOBAUX	TEMPS CPU CONSOMMES SOUS	
			faible charge	forte charge
1	6	79	585 ms	679 ms
2	3	74	469	529
3	82	66	313	431
4	81	62	310	356
5	4	300	1331	2145
6	20	431	862	1003

Figure 5-2: Temps CPU consommés sous différentes charges système.

5.3.4. L'existence de plusieurs "classes" de programmes.

D'autres paramètres de la charge doivent influencer la consommation en temps CPU du LINK ! Par exemple:

- le nombre de modules objets,
- le nombre de modules à rechercher en librairie,
- l'existence de grandes structures de données (grandes matrices),
- le nombre et la complexité des expressions à évaluer[2],
- etc ...

L'association de tels paramètres détermine plusieurs "classes" de programmes dans la charge de travail du LINK.

Il n'est pas possible de collecter toutes ces informations avec l'outil utilisé jusqu'à présent. Cela est dû à la complexité et au grand nombre des informations nécessaires.

Nous avons cependant modifié l'outil utilisé pour qu'il échantillonne également la valeur de la variable CTYPE. Cette variable désigne le langage source du programme principal qui vient d'être traité par le LINK ("MAIN

---

[2]En effet, des expressions faisant intervenir des symboles globaux ne peuvent pas être evaluées lors de la compilation. Le LINK est capable de les évaluer.

COMPILER TYPE"). En effet, certains éléments peuvent varier d'un langage à l'autre :

- le domaine d'utilisation,
- la manière de l'utiliser (du point de vue de l'utilisation de sous-programmes externes, de grandes matrices, d'expressions faisant intervenir des symboles globaux, etc ...),
- certaines particularités dans l'implémantation du langage (Pour certains langages, le LINK doit effectuer des traitements particuliers tels que les "COMMONS" pour le FORTRAN, les "ALGOL OWN BLOCKS" pou l'ALGOL, les "COBOL SYMBOLS", etc...).
- etc...

Le tableau de la figure 5-3. donne la composition de la charge du LINK en fonction du langage source du programme principal. Durant la période considérée, les programmes FORTRAN et COBOL-74 interviennent, respectivement, pour 41% et 42% dans cette charge[3] .

De plus, un tri des observations par langage a permis de réaliser les graphiques décrits en 5.3.1., pour les langages les plus abondamment utilisés (cf. annexe 3.3.).

On remarque ainsi que le COBOL-74 a un comportement assez stable. Par contre, les graphiques des observations de programmes FORTRAN présentent toujours le phénomène "de fourche" dont il a été question.

Une analyse approfondie des programmes FORTRAN peut, sans doute, apporter des résultats intéressants. Par manque de temps, une telle analyse n'a pas été entreprise.

---

[3] D'importantes variations peuvent être observées d'une période à l'autre. Ceci est, principalement, dû au fait que la charge du LINK est composée en grande partie de travaux pratiques d'étudiants. Ces travaux pratiques ne sont pas répartis de manière régulière tout au long de l'année académique.

LANGAGE	NOMBRE D'OCCURRENCES	TEMPS CPU TOTAL (en secondes)
COBOL-68	80	108
ALGOL	391	412
FORTRAN	4209	6238
MACRO	60	20
SIMULA	71	274
COBOL-74	4034	3187
TOTAUX	8845	10239

Figure 5-3: Composition de la charge du LINK en fonction du langage du programme principal.

6. MESURES SUR LE LINK VU COMME UNE "BOITE BLANCHE".

6.1. La méthode.

Ces mesures ont pour but de déterminer où dans son exécution le LINK consomme du temps CPU. Le principe de base est que c'est dans les "parties coûteuses" qu'une amélioration éventuelle s'avèrerait la plus avantageuse.

Le mot "partie" nécessite cependant quelques précisions.

Ces parties ne peuvent pas être les différents modules fonctionnels du LINK car cette découpe est trop grossière (cf. Première Partie, 2.3.1.).

Utiliser, comme parties, les diverses sous-routines du LINK est déjà plus intéressant. Malheureusement, vu le nombre important et la diversité des points de sortie de ces sous-routines, cette possibilité est difficilement réalisable.

Une méthode classique consiste à observer le compteur ordinal (ou "PROGRAM-COUNT" ou "P-COUNT") du programme durant son exécution, afin de construire un histogramme de ses valeurs (cf. [JC], [VF], [JH]).

Une "pointe" sur un tel histogramme indique une zone d'activité intense dans le programme analysé. Ces zones sont les parties coûteuses qui nous intéressent.

Il faudra encore identifier ces zones. Ceci peut se faire grâce à l'utilisation de la "carte" d'implantation du LINK en mémoire (cf. Première Partie, 2.2.2.).

Enfin, une analyse critique du code de ces zones coûteuses permettra, éventuellement, de proposer une amélioration.

Un avantage évident de cette méthode est que l'on peut se permettre de

n'étudier du programme, que ce qui est vraiment important.

On se rend compte, d'autre part, que de telles mesures ne peuvent être réalisées qu'en utilisant un échantillon de la charge réelle du LINK.

6.2. Le choix d'un échantillon de charge.

Les résultats de la première phase des mesures nous fournissent une caractérisation de la charge de travail du LINK. Précisons que cette caractérisation n'est valable que pour l'environnement universitaire dans lequel les mesures ont été réalisées (et pour une période donnée).

Pour pouvoir réaliser des mesures plus détaillées sur la consommation en temps CPU du LINK, nous avons besoin d'une série de programmes issus de cette charge.

Il a déjà été question de l'existence de différentes "classes" de programmes. Théoriquement, il suffirait de choisir, dans chaque classe, un certain nombre de programmes (proportionnellement à la taille de la classe) pour obtenir un échantillon représentatif de la charge réelle.

Pour déterminer ces classes, on peut faire appel à des techniques dites de "clustering" (cf. [DF]). Les observations étant représentées par des points dans l'espace des paramètres, un nuage de points ("CLUSTER") correspond à une classe.

Pourtant, d'après les remarques faites précédemment, nous savons que l'appartenance d'un programme à l'une ou l'autre classe ne peut pas se décider sur base des seuls paramètres connus (taille, nombre de symboles, langage et temps CPU).

Nous avons préféré rechercher quelques programmes en tenant compte des faits suivants :

- 41% de la charge sont constitués par des programmes FORTRAN et 48% sont constitués de programmes COBOL-74.

- 75% des programmes FORTRAN ont une taille inférieure à 100 pages. La quasi totalité des programmes COBOL-74 a une taille inférieure à 100 pages.

- 90% des programmes FORTRAN ont moins de 200 symboles globaux. Plus de 30% des programmes COBOL-74 ont exactement 324 symboles globaux (le reste a entre 324 et 524 symboles globaux).

- Les programmes FORTRAN pour lesquels une amélioration du LINK est la plus avantageuse, sont ceux de la branche supérieure du graphique tailles/temps. En effet, malgré leurs tailles parfois assez réduites (+ou- 30 pages), ils nécessitent beaucoup de temps CPU pour être reliés et chargés.

C'est sur la base de ces remarques que nous avons récolté quelques programmes FORTRAN et COBOL-74[4]. L'échantillon ainsi constitué n'est pas absolument représentatif de la charge réelle.

Le tableau de la figure 6-1 donne la composition de notre échantillon de charge.

LANGAGE	NUMERO	TAILLE (en pages)	NB DE SYMBOLES GLOBAUX	TEMPS CPU NECESSITES (en ms.)
FORTRAN	1	33	70	698
	2	35	98	1196
	3	9	122	2115
	4	3	74	640
	5	22	184	2501
	6	33	214	3863
	7	62	204	4246
	8	6	79	713
COBOL-74	9	87	340	3967
	10	62	411	2307
	11	15	324	889
	12	28	333	1610
	13	6	324	633
	14	40	516	1837
	15	65	466	2580

Figure 6-1: Composition de l'échantillon de charge.

---

[4]Pour ce faire, nous nous sommes adressés aux personnes les plus susceptibles de nous fournir les programmes adéquats

6.3. L'outil disponible : LOOK.

6.3.1. Description de LOOK.

LOOK est un programme fourni (mais non supporté) par la firme Digital Equipment Corporation. Il permet d'"observer" n'importe quel autre programme. Cette observation (ou "MONITORING") consiste en un échantillonnage des valeurs du compteur ordinal du programme analysé.

Pour ce faire, LOOK considère l'espace d'adressage du programme comme divisé en un certain nombre de zones de 8 mots. Il maintient une table de compteurs. Chaque fois que le compteur ordinal est échantillonné, la valeur obtenue permet de déterminer le compteur à incrémenter.

A la fin de l'observation, LOOK construit un histogramme à partir de la table de compteurs.

Le fonctionnement de LOOK peut être résumé comme suit :

1. LOOK s'exécute évidemment dans un processus. A la demande de l'utilisateur, LOOK crée un processus-fils dans lequel il place le programme à observer.
2. Il initialise la table des compteurs.
3. Il lance l'exécution du programme.
4. Il se met en attente pour un temps paramétrable (il s'agit d'un intervalle de temps-réel ou temps-écoulé).
5. Lorsque le temps désiré s'est écoulé, LOOK est "réveillé" par le gérant des processus. A ce moment, LOOK demande, à ce gérant des processus, le mot d'état du processus-fils. Ce mot d'état fourni l'état du processus et la valeur du compteur ordinal.
6. Si l'état du processus-file est "EN EXECUTION", LOOK incrémente le compteur correspondant à la zone de 8 mots calculée à partir de la valeur du compteur ordinal. Ensuite, LOOK recommence en 4.
7. Lorsque l'exécution du programme est terminée, l'observation est interrompue. L'utilisateur peut alors demander que l'histogramme résultant lui soit fournit soit au terminal, soit sur un fichier.

6.3.2. Premières mesures et critiques de l'outil.

Les premières mesures effectuées avec LOOK font apparaître divers problèmes.

6.3.2.1. Le nombre d'échantillonnages du compteur ordinal.

Le nombre d'échantillonages du compteur ordinal est très réduit. Ceci provient du fait que, d'une part, les programmes qui constituent la charge du LINK lors de ces mesures ne sont pas très grands (Le temps de réponse du LINK est de quelques secondes) et d'autre part, l'intervalle d'échantillonnage par défaut est de 100 millisecondes.

Dans le but d'augmenter le nombre d'observations, on peut vouloir diminuer la longueur de l'intervalle d'échantillonnage. On se heurte alors aux limitations du système d'exploitation TOPS-20. En effet, le cycle du gérant des processus ("SCHEDULER") est de 20 millisecondes ou, autrement dit, l'intervalle minimal entre deux interventions du "SCHEDULER" est de 20 millisecondes. De plus, le processus dans lequel LOOK s'exécute devra encore attendre dans la file d'attente des processus prêts. Par conséquent, l'intervalle d'échantillonnage n'est pas une constante mais obéit plutôt à une loi de probabilité.

Pratiquement, ceci se traduit par de fortes variations dans la forme des histogrammes provenant d'expériences successives.

6.3.2.2. Le cumul des résultats.

Il n'est pas possible de cumuler les résultats de plusieurs mesures successives (avec toujours le même programme en entrée au LINK), dans l'espoir d'arriver à des résultats plus stables. Ceci s'explique par le fait que le LINK se détruit lui-même en construisant le programme exécutable. En effet,

lorsque l'on veut recommencer l'expérience, il faut recharger la version exécutable du LINK dans l'espace d'adressage du processus-fils. Ceci se fait grâce à la commande "GET". L'utilisation de celle-ci remet à zéro la table des compteurs.

6.3.2.3. PA1050.

Une proportion importante des observations se situent dans l'espace occupé par PA1050 (cf. Première Partie, 2.4.). Pour certains programmes de l'échantillon, cette proportion peut aller jusqu'à 80% des observations ! Les résultats de LOOK ne fournissent donc pas beaucoup de renseignements sur l'exécution du LINK.

6.4. Mise au point d'un outil propre.

Les remarques précédentes ont guidé la mise au point d'un outil propre (MYLOOK). Celui-ci, bien que basé sur le même principe que LOOK, permet la répétition automatique des expériences avec cumul des résultats. A la fin de la dernière expérience, l'outil fournit l'histogramme désiré.

Le nombre de répétitions et l'intervalle d'échantillonnage sont paramétrables.

De plus, toute observation faite dans l'espace occupé par PA1050 (pages 700b à 730b de l'espace d'adressage) peut être imputée à la zone où se trouve l'appel-système qui a causé le déroutement.

Cependant, le fait de pouvoir répéter l'expérience un certain nombre de fois, introduit une question importante : combien de fois ?

Actuellement, nous ne pouvons pas répondre à cette question avec exactitude. Il faut essayer avec 50, 100 ou 200 répétitions et choisir le résultat le plus satisfaisant (du point de vue du nombre d'observations et de la stabilité). D'autant plus que les résultats dépendent fortement des programmes introduits en input au LINK. De plus, il faut également tâtonner

pour déterminer l'intervalle d'échantillonnage à utiliser.

Cependant, notre but n'est pas de construire un outil parfait. L'outil nous fournit ce dont nous avons besoin : des indications quant aux endroits où une investigation plus poussée peut s'avérer utile (Il s'agit, avant tout, d'une question de besoins).

6.5. Les résultats.

Après quelques essais, les résultats obtenus sont assez satisfaisants. Ils ont été réalisés en répétant 40 fois l'exécution du LINK avec le même programme en entrée. L'intervalle d'échantillonnage était de 40 millisecondes.

Le rapport fournit :

- le nombre de répétitions,
- le temps CPU moyen consommé pour une édition de liens-chargement,
- le nombre de fois que le compteur ordinal a été échantillonné,
- le détail des observations par état ("RUN", "I/O-WAIT",...),
- L'histogramme des valeurs du compteur ordinal.

(Voir annexe 4. pour le programme et les résultats.)

TROISIEME PARTIE.

EXPLOITATION DES RESULTATS DES MESURES DE PERFORMANCES.

7. INTRODUCTION.

Nous allons traiter un seul exemple entièrement pour mettre en évidence ce qui peut être fait une fois que les mesures ont été achevées. Dans la suite, nous citerons d'autres exemples sans les traiter de manière aussi détaillée.

La démarche est la suivante :

- localisation d'une zone coûteuse dans le programme LINK,
- identification de la zone,
- description du code dans cette zone,
- explication du fait que ce code est coûteux en temps CPU,
- propositions d'améliorations du code (amélioration "interne") ou de son utilisation (amélioration "externe"),
- implémentation de ces propositions,
- nouvelles mesures.

8. UN EXEMPLE.

8.1. Localisation d'une zone coûteuse dans le LINK.

Les histogrammes fournis par MYLOOK constituent des "profils" d'exécution du LINK pour chacun des programmes formant la charge-test.

Ces profils d'exécution ne donnent pas avec une exactitude absolue les proportions du temps CPU passé dans les différentes parties du LINK (ceci est dû à la méthode utilisée). Cependant, ils donnent des estimations de ces proportions (un certain nombre d'observations du compteur ordinal du LINK durant son exécution, se situent dans une zone donnée de 8 mots de l'espace d'adressage).

Ces histogrammes révèlent une pointe très importante dans la zone 564610b de l'espace d'adressage (adresses 564610b à 564617b). La proportion des observations faites dans cette zone varie beaucoup avec le type de programme en entrée au LINK (cf. annexe 4.2). Pour des petits programmes FORTRAN, cette proportion représente jusqu'à 25% du total des observations du compteur ordinal. Pour les programmes COBOL-74, elle est moins importante (+ ou - 10%).

Par conséquent, une analyse approfondie du code dans cette zone peut être utile.

8.2. Identification de la zone.

A l'aide de la "carte" d'implantation du LINK en mémoire (cf. Première Partie, 2.2.2.), nous pouvons voir que :

- la zone en question se trouve dans le module LNKOLD,
- le symbole NXTBLK correspond à l'adresse 564610b, c'est-à-dire l'adresse d'un mot dans la zone en question.

Ces indications permettent de trouver le code correspondant dans le programme-source du LINK. Il s'agit, plus précisément, de la procédure T.14 qui assure le traitement des REL BLOCKS de type 14 c'est-à-dire des blocs d'index de librairies.

8.3. Description des mécanismes en cause.

8.3.1. Description des blocs d'index.

Les blocs d'index sont utilisés pour accélérer la recherche en librairie.

Un bloc d'index est constitué de 128 mots. Il comporte un certain nombre de sous-blocs (cf. figures 8-1 et 8-2). Pour chacun des modules relogeables qui suivent, il y a un sous-bloc. Ce dernier contient la liste des points d'entrées ("ENTRY POINTS") du module correspondant ainsi qu'un pointeur vers celui-ci.

Si un bloc d'index ne suffit pas, on peut chaîner un nombre quelconque d'entre eux. Le dernier mot d'un bloc d'index contient soit un pointeur vers le suivant, soit -1 pour signaler qu'il s'agit du dernier (cf. figure 8-3).

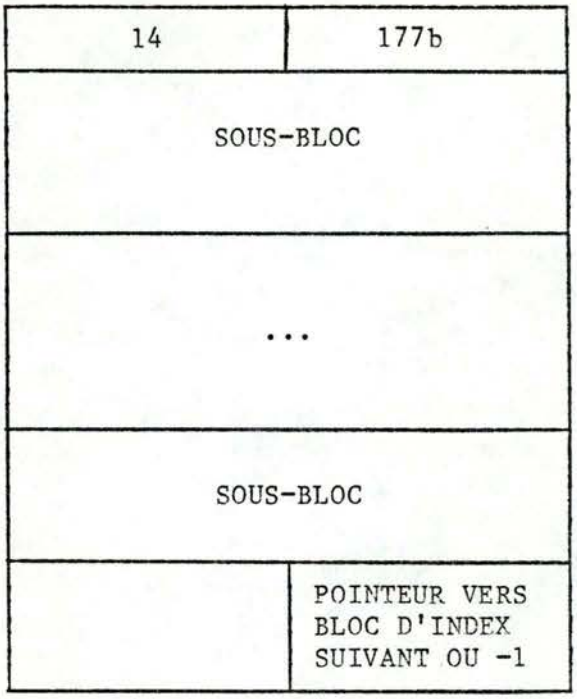


Figure 8-1: Structure d'un REL BLOCK de type 14.

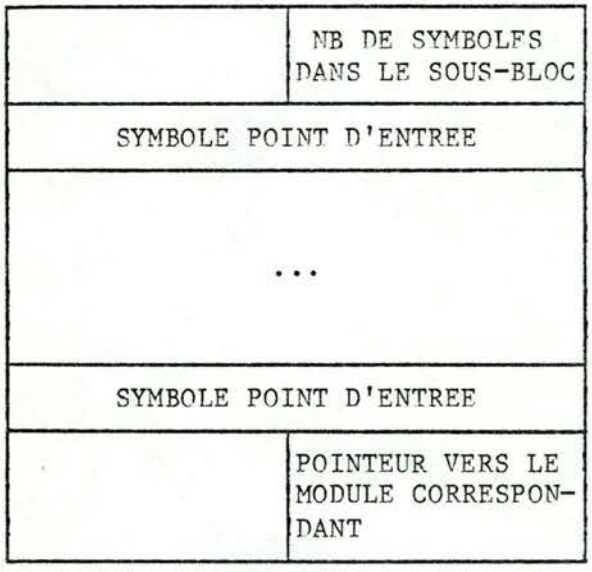


Figure 8-2: Structure d'un sous-bloc.

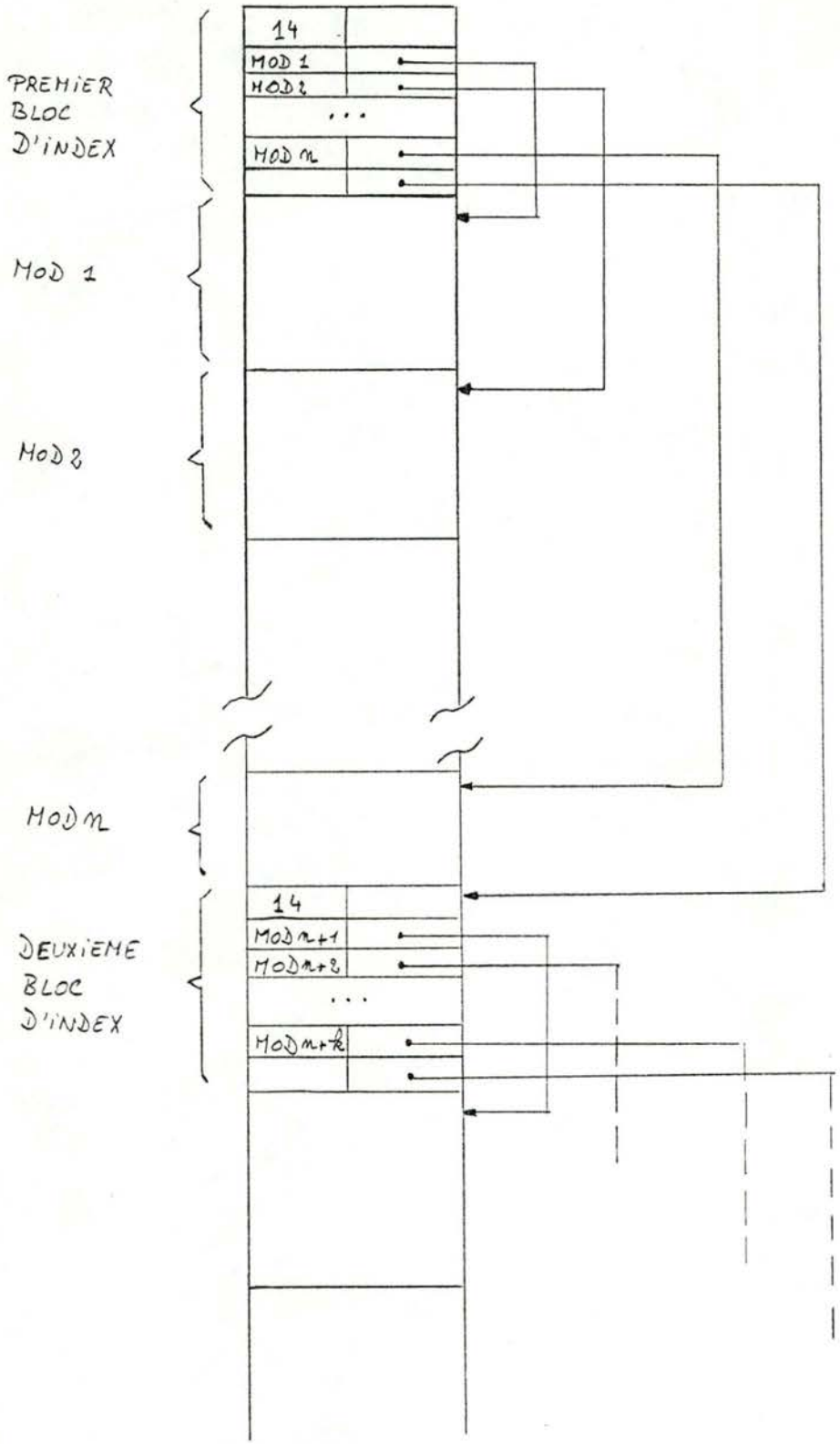


Figure 8-3: Structure d'une librairie indexée.

8.3.2. Description de la procédure T.14.

D'une manière générale, la lecture d'un "HEADER WORD" (cf. Première Partie, 2.1.1.) permet de savoir de quel type de bloc il s'agit. D'après le type de bloc, le LINK détermine le traitement (la procédure) à appliquer au reste du bloc.

Ayant ainsi déterminé qu'un bloc est de type 14, le LINK poursuit son exécution en T.14. Cette procédure parcourt séquentiellement le bloc d'index et le traitement peut être résumé comme suit :

1. S'il s'agit du premier bloc d'index rencontré, le LINK demande un bloc de travail de 128 mots dans la zone d'allocation dynamique de mémoire.

2. Le bloc d'index est recopié dans le bloc de travail.

3. Le LINK lit un symbole (point d'entrée) dans le bloc de travail. (chaque point d'entrée est codé en code RADIX-50. Il doit, d'abord, être traduit en code SIXBIT). Le symbole est alors recherché dans la table des symboles globaux.

4. Si le symbole ne se trouve pas dans la table ou s'il s'y trouve mais qu'il est déjà défini, alors on poursuit en 3.

   Sinon (c'est-à-dire si le symbole se trouve dans la table sans avoir été défini jusqu'ici) il constitue une requête globale non encore résolue. Dans ce cas, le module correspondant doit être chargé. Pour ce faire, on se positionne dans le fichier en utilisant le pointeur fourni dans le sous-bloc courant. Ensuite, le chargement se fait comme pour tout autre module.

5. Lorsque le module est chargé et s'il reste des symboles globaux non définis, on continue en 6. Sinon, la recherche en librairie est terminée.

6. Eventuellement, il reste encore des symboles dans le sous-bloc courant. Il sont ignorés puisque le module correspondant vient quand même d'être chargé.

   S'il reste encore des sous-blocs à considérer dans le bloc d'index courant, on retourne en 3.

   Sinon, deux cas peuvent se présenter :

   - le bloc d'index qui vient d'être lu est le dernier de la librairie. Dans ce cas, la recherche en librairie est terminée.

   - le bloc d'index qui vient d'être lu n'est pas le dernier. Il y a lieu d'utiliser le pointeur fourni en fin de bloc pour se

positionner dans le fichier et lire le bloc d'index suivant. Lorsque ceci est fait, on retourne en 2.

Le détail du code de la procédure T.14 est fournit en annexe 5.1.

8.3.3. Description succincte de la lecture d'un fichier sous TOPS-10.

La zone qui nous intéresse correspond aux instructions qui réalisent la lecture du fichier-librairie sur disque. Dès lors, il est intéressant de décrire le déroulement de cette lecture. Ensuite, nous pourrons expliquer pourquoi ce code coûte cher en temps CPU.

Le LINK utilise évidemment la technique propre à TOPS-10 pour lire un fichier (cf. Première-Partie, 2.3.3.). Ceci est vrai aussi bien pour les fichiers objets des utilisateurs que pour les librairies.

Essentiellement, les opérations à exécuter par un programme désirant lire un fichier sur disque sont les suivantes (cf. [MC10], p. 12-1) :

- Initialiser un canal d'entrée/sortie (le rendre disponible pour le programme),
- Initialiser un anneau de tampons ("BUFFER RING"),
- Sélectionner un fichier,
- Faire remplir les tampons par le système d'exploitation et traiter les données ainsi lues,
- Fermer le fichier,
- Libérer le canal.

Le fait de travailler avec un anneau de tampons doit permettre la simultanéité entre traitement des données par le programme et le transfert physique des données. Pendant que le programme lit ou écrit dans un tampon, le transfert physique se fait de ou vers le ou les autre(s) tampon(s).

La taille de chaque tampon est de 128 mots i.e. la taille d'un bloc de données sur disque ("DISK-BLOCK").

La figure 8-4. montre la structure d'un anneau de 3 tampons.

Dans le "BUFFER CONTROL BLOCK" se trouvent, principalement, un pointeur vers le tampon courant, un pointeur vers le dernier byte lu et le nombre de bytes restant à lire dans ce tampon.

Les "BUFFER HEADER BLOCKS" contiennent, entre autres, un pointeur vers le tampon suivant dans l'anneau.

Ces informations de service sont tenues à jour soit par le programme, soit par le système d'exploitation.

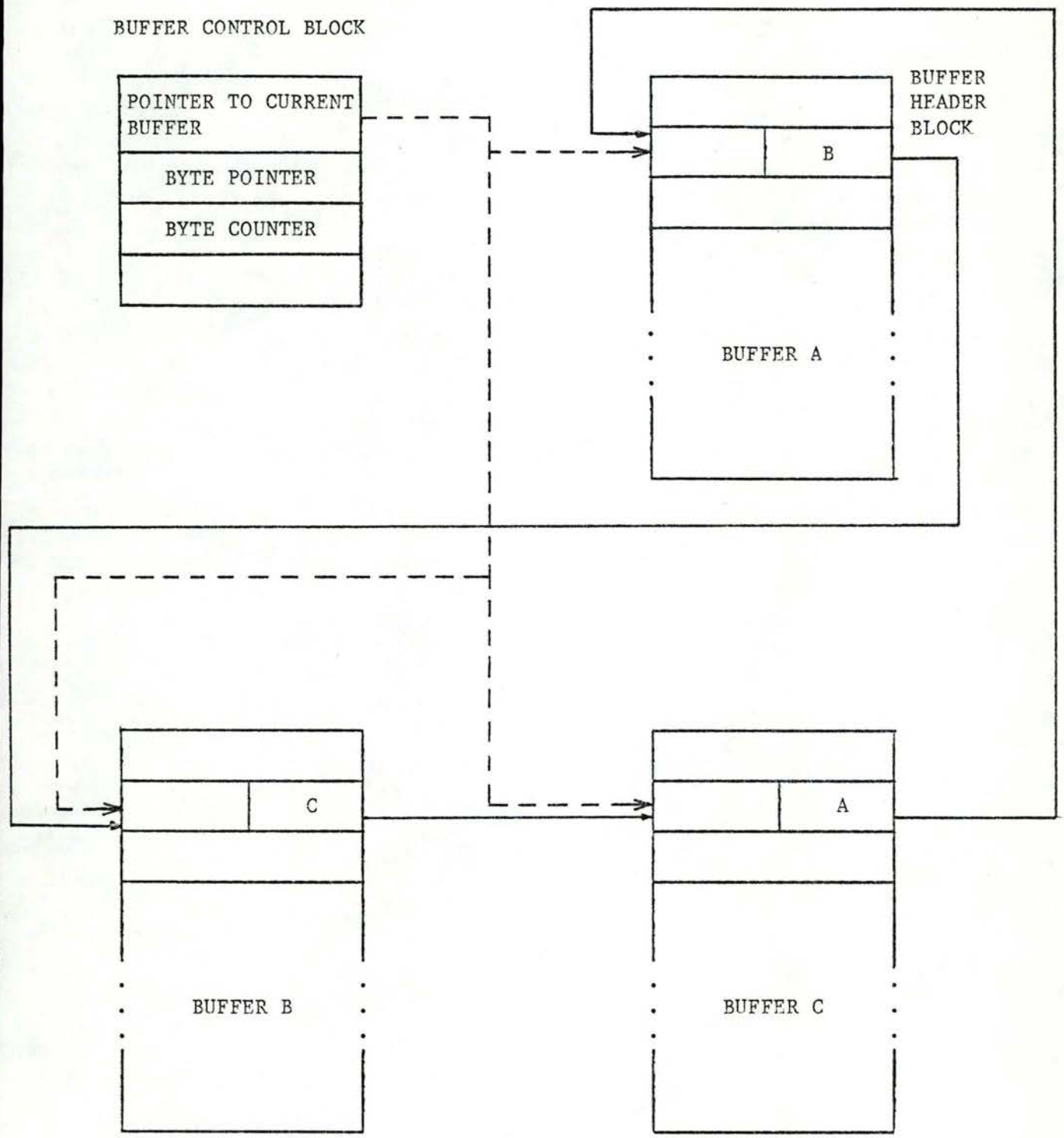


Figure 8-4: Structure d'un anneau de 3 tampons.

8.4. Pourquoi ce code est-il coûteux?

La pointe localisée dans les profils d'exécution du LINK, correspond à l'appel-système qui demande le passage au tampon suivant dans l'anneau et donc la lecture d'un "DISK BLOCK" (IN UUO). Le tampon qui vient d'être lu peut être rempli à nouveau par le système d'exploitation. Dans ce cas particulier, on s'est préalablement positionné dans le fichier-librairie de manière à ce que le bloc lu soit le prochain bloc d'index.

Comme nous l'avons déjà fait remarquer, le LINK utilise les appels-systèmes propres au système d'exploitation TOPS-10. Dès lors, son exécution sous le système d'exploitation TOPS-20 nécessite l'interprétation des appels-systèmes par l'interface PA1050 (cf. Première-Partie, 2.3.3.).

PA1050 utilise la technique propre à TOPS-20 pour la lecture d'un fichier. Il possède ses propres tampons. L'interprétation de l'appel-système IN consiste en un recopiage du contenu des tampons de PA1050 dans les tampons du LINK. De plus, PA1050 doit assurer la mise à jour des informations de service contenues dans le "BUFFER CONTROL BLOCK" et dans les "BUFFER HEADER BLOCKS". Il n'y a plus de simultanéité entre remplissage des tampons et traitement des données par le programme !

Cette recopie des données d'un tampon dans l'autre est coûteuse en temps CPU. La preuve en est que les histogrammes construits par LOOK ou MYLOOK et qui tiennent compte de PA1050 (cf. annexe 4.2.1.) présentent tous une pointe extrêmement importante à l'endroit de l'instruction BLT ("BLOCK TRANSFERT") qui assure ce recopiage (zone 704350b de l'espace d'adressage).

8.5. Amélioration "interne".

8.5.1. Amélioration par adaptation.

Une première possibilité d'améliorer les performances du LINK consiste à adapter celui-ci au système d'exploitation TOPS-20. La lecture de tous les fichiers (pas uniquement les fichiers-librairies) pourrait être accélérée si le LINK utilisait directement les appels-systèmes propres à TOPS-20, sans passer par PA1050.

8.5.2. Description succincte de la lecture d'un fichier sous TOPS-20.

La technique utilisée par PA1050 est appelée "PAGE MAPPING" (cf. [MC20], [MO]). Elle consiste à rendre équivalentes une ou plusieurs pages de l'espace d'adressage du processus (le tampon) et autant de pages du fichier que l'on veut lire.

Cette équivalence ne cause aucun transfert physique de données et consiste, en fait, en une modification de la table des pages du processus. Cette table indique où se trouvent les pages existantes du processus : en mémoire centrale ou en mémoire auxiliaire. Ce n'est que lorsque le programme fait référence à ces pages (pour lire le contenu du fichier) et qu'elles ne se trouvent pas en mémoire centrale, qu'il y a un défaut de page qui entraîne le transfert physique.

Ces pages constituent une "fenêtre" à travers laquelle le programme lit le fichier. On peut déplacer cette fenêtre progressivement, de manière à lire tout le fichier (cf. figure 8-5).

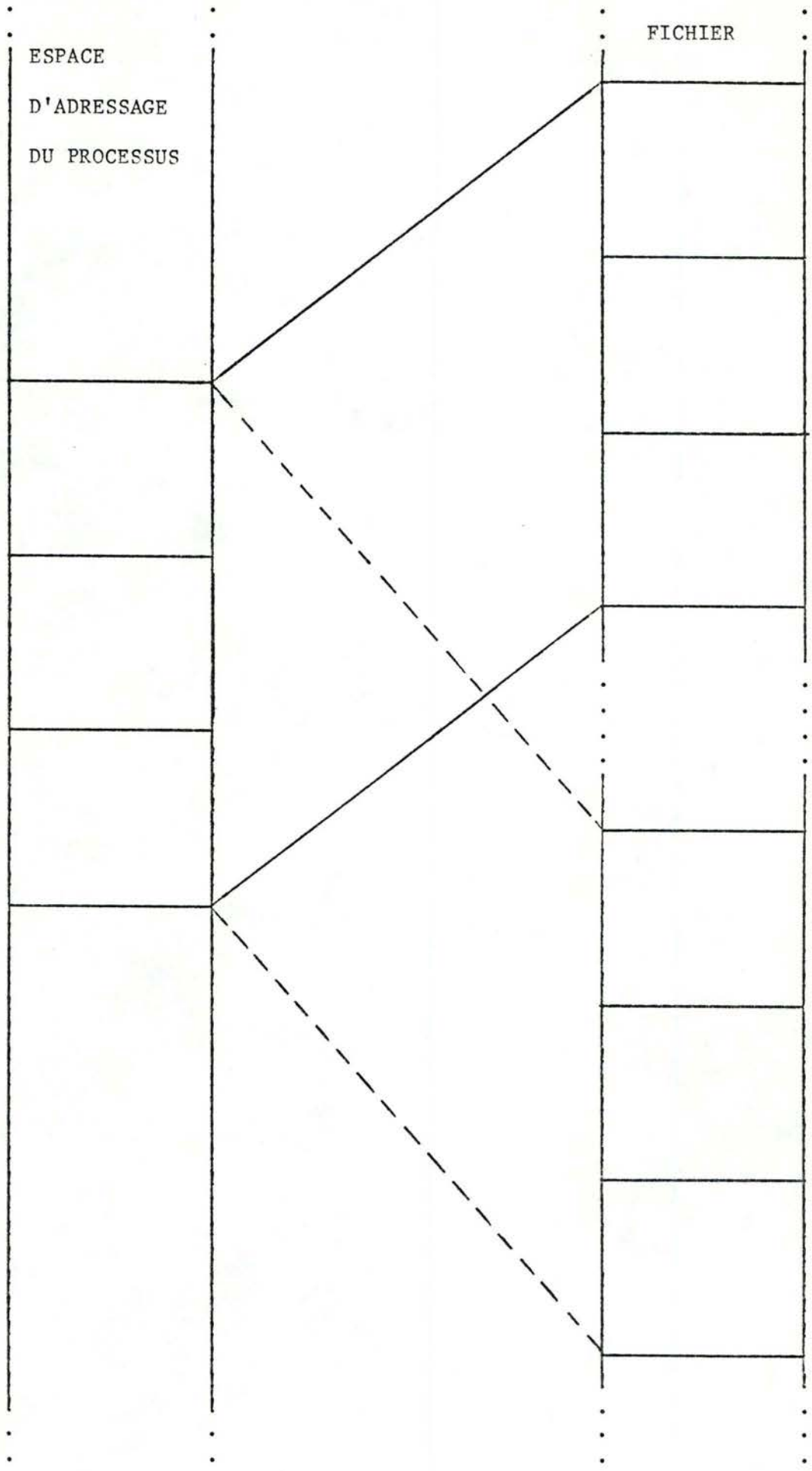


Figure 8-5: Le "PAGE MAPPING".

8.5.3. L'implémentation.

Il s'agit d'une modification d'une importance telle qu'une implémentation est irréalisable dans le cadre de ce mémoire.

A défaut de pouvoir réaliser cette adaptation du LINK, nous avons cependant essayé de montrer qu'il s'agit effectivement d'une amélioration. Pour ce faire, nous avons écrit deux programmes.

BFRDIO — qui réalise la lecture d'un fichier en utilisant les appels-systèmes TOPS-10 (et nécessite, par conséquent, PA1050 pour l'interprétation de ces appels).

PMAPIO — qui réalise la lecture d'un fichier en utilisant les appels-systèmes TOPS-20 (et donc la technique du "PAGE MAPPING").

Nous avons, ensuite, mesuré le temps CPU consommé par ces deux programmes pour lire les mêmes fichiers. Les résultats sont probants : PMAPIO consomme jusqu'à 45% de moins que BFRDIO ! (pour le détail des programmes et des résultats, voir annexe 5.2.).

8.6. Amélioration "externe".

8.6.1. Le principe.

La première possibilité consistait à modifier le LINK lui-même (amélioration "interne"). Une seconde possibilité peut être de diminuer le nombre de lectures sur fichiers que le LINK doit exécuter.

Il n'est évidemment pas possible de diminuer le nombre de lectures nécessaires pour les fichiers objets de l'utilisateur. Ces fichiers doivent être parcourus entièrement. Cependant, l'endroit coûteux que nous avons identifié concerne le traitement des librairies. Or une réorganisation judicieuse de ces librairies peut, dans certains cas, apporter une diminution notable des lectures sur disque et du temps CPU consommé (cf. [BCM]). Par opposition à la première possibilité, on peut parler d'une amélioration "externe".

8.6.2. L'implémentation.

Une analyse plus poussée serait nécessaire pour pouvoir mettre en évidence, pour chacune des librairies-systèmes, un "noyau" de modules qui sont souvent (si pas toujours) référencés. L'outil utilisé pour les mesures sur le LINK vu comme une boîte noire (cf. Deuxième Partie 6.2.) pourrait être adapté de manière à collecter les noms des modules recherchés dans ces librairies.

Ensuite, il faut encore tenir compte des dépendances éventuelles entre certains modules d'une librairie donnée. L'ordre des modules peut avoir de l'importance. En effet, une librairie est parcourue sans retour en arrière. Dès lors, si un module fait référence à un autre module de la même librairie, ce dernier doit nécessairement venir après le premier.

Finalement, le noyau ainsi déterminé peut être placé au début de la

librairie (les modules étant, dans la mesure du possible, classés par ordre de fréquences d'utilisations décroissantes). De cette façon, la recherche en librairie peut être accélérée.

Une première implémentation est représentée à la figure 8-6. Les modules du noyau y sont accessibles via le premier ou le deuxième bloc d'index. Cette solution est réalisable grâce à MAKLIB. Cet utilitaire permet diverses manipulations sur une librairie : ajouter, enlever ou déplacer un module, créer un index, etc... (cf. [MUG]).

Une seconde implémentation consiste à placer les quelques modules qui sont toujours référencés, en tête de la librairie, avant le premier bloc d'index (cf. figure 8-7). Ceci permettrait d'aller encore plus vite. Malheureusement, MAKLIB ne permet pas ce genre de modifications (Dans une librairie indexée, le premier REL BLOCK est nécessairement un bloc d'index. Il s'agit d'une limitation de MAKLIB).

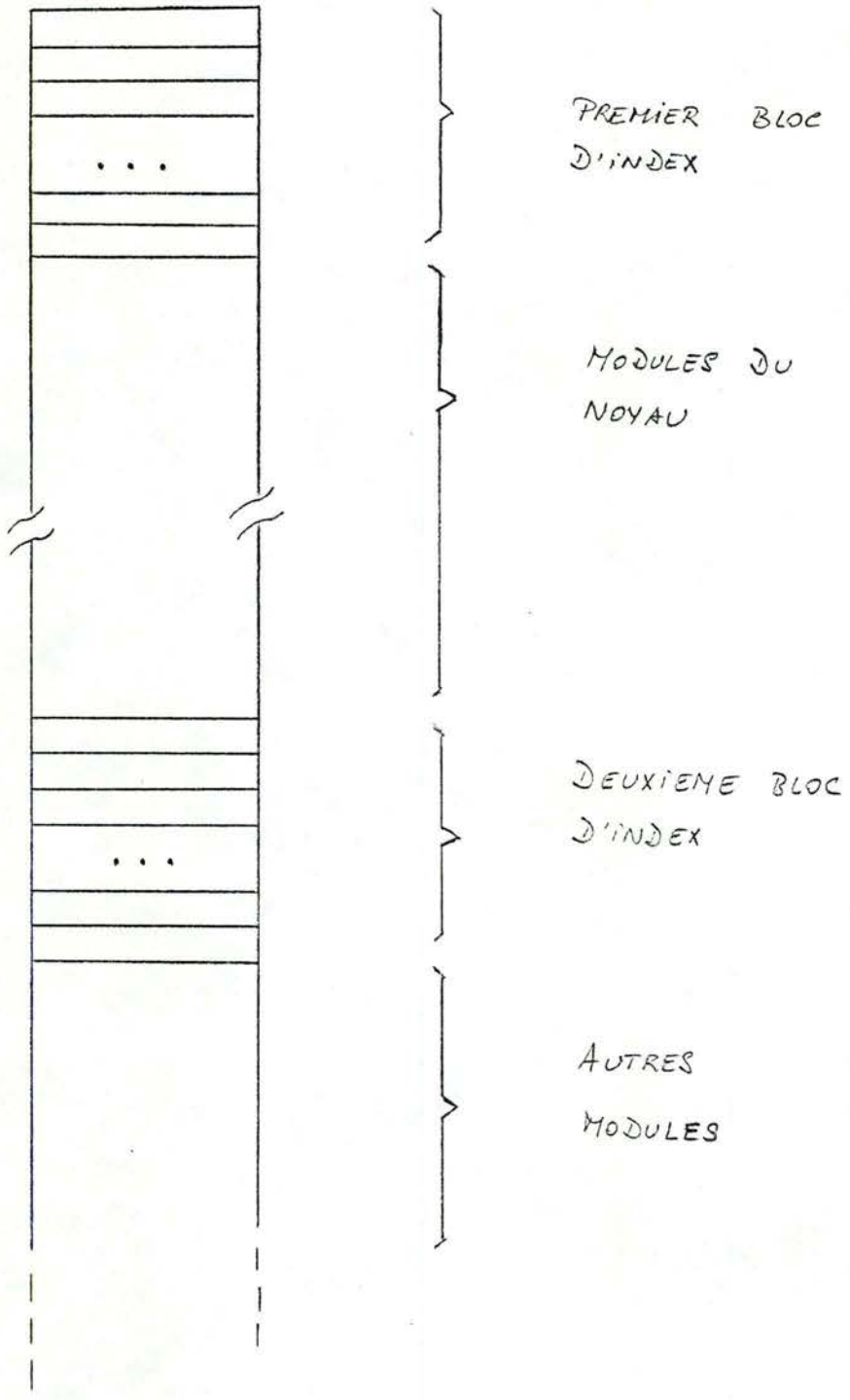


Figure 8-6: Première possibilité de réorganisation d'une librairie indexée.

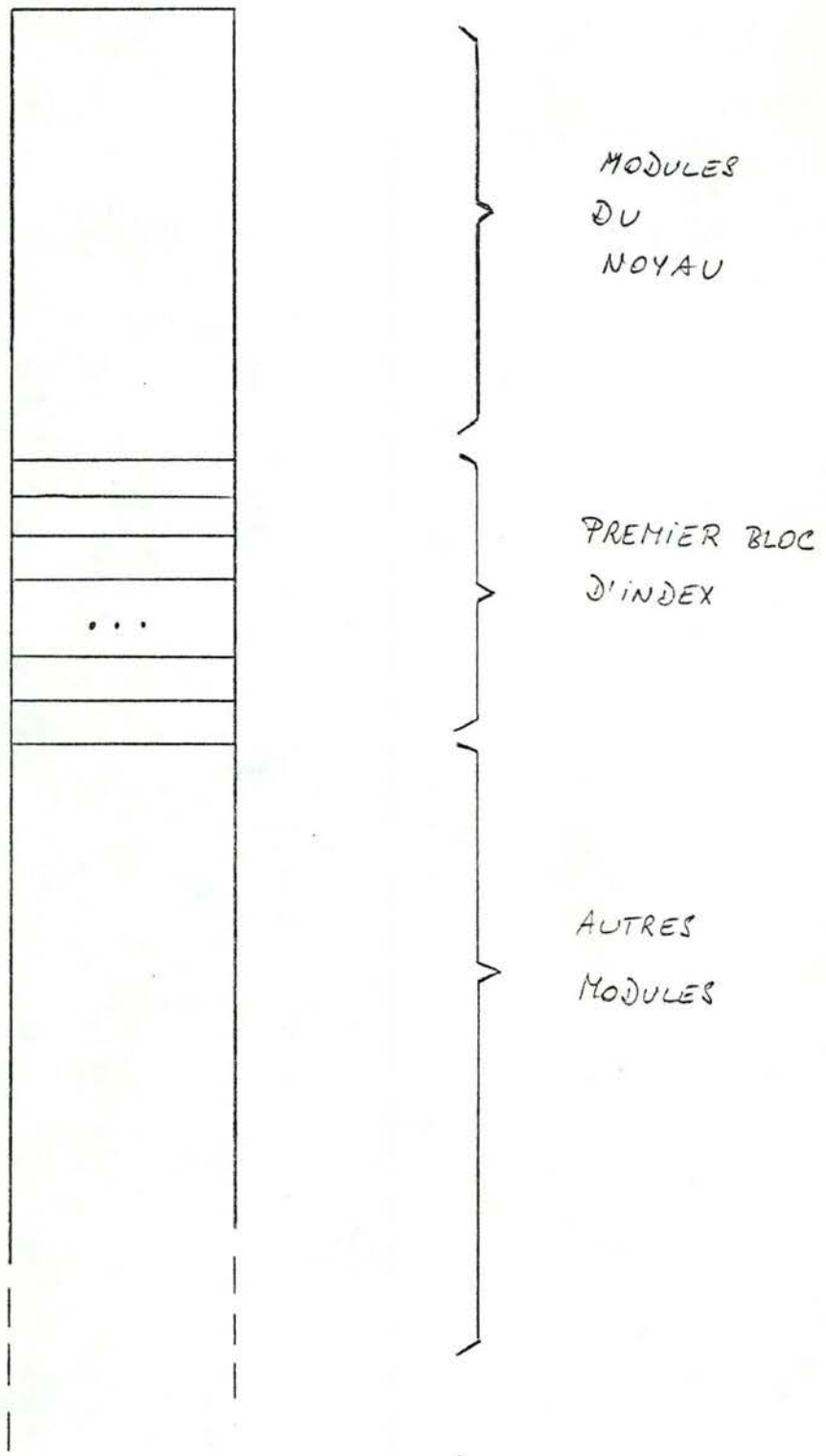


Figure 8-7: Deuxième possibilité de réorganisation d'une librairie indexée.

8.6.3. Les nouvelles mesures.

Par manque de temps, les mesures préconisées ci-dessus n'ont pas été réalisées. Nous avons cependant analysé les programmes FORTRAN de notre échantillon. On peut ainsi remarquer que 3 modules de la librairie FORLIB sont toujours chargés. Il s'agit des modules FORINI, FORPSE et CFRXIT.

Les modules FORINI et FORPSE sont accessibles grâce au premier bloc d'index. Ils sont donc trouvés assez rapidement. Par contre, le module CFRXIT se situe à la fin de la librairie. Par conséquent, le LINK doit parcourir les 8 blocs d'index de la librairie FORLIB avant de trouver ce dernier module.

Avec l'aide de MAKLIB, nous avons modifié la librairie FORLIB en plaçant les 3 modules FORINI, FORPSE et CFRXIT au début (voir annexe 5.3. pour le détail des manipulations). Ensuite, les programmes FORTRAN de l'échantillon ont été reliés et chargés en utilisant cette nouvelle version de FORLIB.

Les résultats montrent qu'il est possible de gagner jusqu'à 19% du temps CPU consommé pour l'édition de liens-chargement de petits programmes. Pour ces derniers, les résultats de nouvelles mesures avec MYLOOK ne présentent plus aucune trace de la pointe en 564610b (cf. annexe 5.4.).

Lorsque d'autres modules doivent être recherchés dans FORLIB, les gains sont moins importants (voir tableau de la figure 8-8). Bien entendu, ces résultats ne sont valables que pour l'échantillon utilisé.

Les temps CPU indiqués dans ce tableau sont des moyennes. Il a fallu tenir compte des variations dues à la charge du système (cf. Deuxième Partie 5.3.3.).

PROGRAMME NUMERO	NB DE MODULES RECHERCHES	TEMPS CPU CONSOMMES (en ms.)		
		AVANT	APRES	GAIN (en %)
1	1	695	695	0
2	9	1160	1010	13
3	5	2115	2115	0
4	4	640	520	19
5	9	2470	2470	0
6	17	3800	3800	0
7	15	4270	4270	0
8	5	713	608	15

Figure 8-8: Résultats des mesures avant et après la réorganisation de la librairie FORLIB.

9. D'AUTRES EXEMPLES.

Nous donnons ici d'autres exemples d'améliorations possibles mais parfois moins importantes que celles présentées dans le chapitre précédent. Ces remarques proviennent soit des mesures réalisées, soit d'une analyse de certaines parties du code du LINK, soit encore de l'analyse de quelques fichiers objets.

9.1. La lecture des fichiers.

Une seconde pointe dans les profils d'exécution du LINK attire notre attention. Elle se situe en 557540b et correspond aussi à la lecture des fichiers objets (du moins en ce qui concerne les REL BLOCKS autres que les blocs d'index).

De plus, une troisième pointe correspond à l'appel-système LOOKUP (en 553120b). Celui-ci sert à sélectionner un fichier pour la lecture (cf. [MC10], p. 27-171). L'interprétation de cet appel est très coûteux en temps CPU car il n'existe rien d'analogue parmi les appels-systèmes TOPS-20.

L'amélioration du LINK par adaptation est donc d'autant plus souhaitable.

9.2. L'initialisation.

Dans le module LNKINI sont réalisées différentes initialisations qui, pour des petits programmes, nécessite jusqu'à 10% du temps CPU total.

Parmi ces initialisations figure une mise à zéro de l'espace occupé par la "base de donnée" du LINK. Or, cet espace est dejà à zéro ! (Le mécanisme de mémoire virtuelle du DECsystem-20 se charge de mettre à zéro toute page nouvellement créée.)

D'autre part, le LINK y effectue une série d'appels-systèmes dont l'utilité est discutable. Par exemple :

- demander le numéro du "JOB". Le LINK ne s'en sert que pour donner un nom à un éventuel fichier .LOG ou .MAP dans le cas où l'utilisateur n'en a pas spécifié,
- demander l'allocation de place mémoire pour les zones de travail. Or, sous TOPS-20, le mécanisme de mémoire virtuelle ne nécessite pas ce genre de demande,
- demander l'heure.

Certains de ces appels-systèmes sont inutiles sous TOPS-20. D'autres pourraient, tout simplement, être exécutés au moment où ils sont nécessaires.

9.3. La lecture des lignes de commandes.

Actuellement, le LINK utilise une série de routines regroupées dans le module .SCAN, pour effectuer la lecture d'une ligne de commandes au terminal. Ce module est d'utilisation généralisée (cf. Premiere Partie 2.3.1.) et est orienté TOPS-10.

Sous TOPS-20, on peut utiliser les facilités offertes par le système (COMND JSYS, cf. [MC20], p. 3-22).

9.4. Les messages d'erreurs du LINK.

Le LINK utilise une manière assez particulière pour signaler des événements (tel que l'entrée dans un module) ou des erreurs. Selon ce que l'utilisateur demande, des messages sont affichés à l'écran et/ou écrits dans un fichier .LOG (cf. Première Partie 2.2.2.).

Notons encore que l'utilisateur peut aussi spécifier quels sont les types de messages qu'il veut voir affichés. En effet, les différents messages sont caractérisés par un niveau et l'utilisateur peut choisir à partir de quel

niveau les messsages doivent être affichés à l'écran. Par défaut, seuls les messages d'erreurs suffisamment graves le sont (cf. [LRM], p. D-12).

Pour générer un message, le LINK exécute une instruction définie lors de l'assemblage du module LNKPAR (l'instruction ERRUUO). Cette instruction possède le code opératoire 001. Celui-ci n'est normalement pas assigné et son exécution résulte en un déroutement vers le système d'exploitation TOPS-20. L'effet de celui-ci est de placer l'instruction ayant causé le déroutement en .JBUUO, et l'adresse de cette instruction +1 en UUOTRAP. Ce déroutement cause enfin l'exécution de l'instruction située en .JB41. En .JB41, le LINK aura mis une instruction de branchement vers une routine qui vérifie s'il s'agit bien d'un code opératoire 001 qui a causé le déroutement. Si ce n'est pas le cas, l'exécution du LINK est interrompue. Par contre, si le test est positif, il y a branchement vers %%UUO. Là, se décide si oui ou non il y a lieu d'afficher le message au terminal et/ou de l'écrire dans le fichier .LOG. Ceci se décide grâce aux informations situées dans le mot suivant l'instruction ERRUUO.

A la fin du traitement, l'exécution du LINK reprend à l'adresse de cette instruction + 2 ! (sauf s'il s'agit d'une erreur grave, auquel cas l'exécution du LINK est interrompue.)

On voit donc qu'il s'agit d'un mécanisme inutilement complexe. Il peut être remplacé, avantageusement, par un mécanisme plus simple du type suivant :

```
Si NIVEAU-MESSAGE > ou = NIVEAU-DEMANDE
        ALORS AFFICHER/ECRIRE MESSAGE
        SINON CONTINUER EN SEQUENCE
```

9.5. Les recherches en librairies.

Il apparaît que, pour certains langages, la librairie-système correspondante est parcourue deux fois ! Il en est ainsi du COBOL-68 et, dans certains cas, du PASCAL.

Dans le cas du SIMULA, le LINK effectue une recherche complexe dans 3 à 4 librairies !

Vu le coût de la recherche en librairie, il y a intérêt à étudier, de manière critique, les raisons de ces recherches.

Dans le cas du COBOL-68, par exemple, le compilateur inclu, dans le fichier objet qu'il produit, un "ASCII TEXT BLOCK". Celui-ci constitue une ligne de commandes pour le LINK. Elle force le LINK à charger le module CON012 de la librairie SYS:LIBOL.REL. Celle-ci est, de toute façon, consultée à la fin du chargement. Il serait intéressant d'étudier une autre possibilité de charger ce module. Eventuellement, l'existence d'un symbole global représentant un point d'entrée dans le module CON012 et non défini lors de la recherche finale en librairie, résulterait dans le chargement de ce module (selon le mécanisme décrit en 2.3.2.).

CONCLUSIONS.

CONCLUSIONS.

Dans le but de pouvoir proposer des améliorations à l'éditeur de liens LINK, nous avons procédé à une étude critique des performances de cet utilitaire. Nous nous sommes cependant limités aux performances du point de vue de la consommation en temps CPU. Une autre possibilité pourrait être, par exemple, l' analyse de la consommation en place mémoire du LINK.

Une part importante de cette étude a été consacrée à la mise au point de deux outils de mesure. Le premier nous a permis de caratériser la charge de travail du LINK selon certains critères (la taille, le nombre de symboles globaux, le langage des programmes ainsi que le temps CPU consommé lors de l'édition de liens-chargement de ceux-ci).

Grâce à cela, nous avons pu déterminer l'échantillon de programmes nécessaire lors des mesures plus précises effectuées avec le second outil. Ces dernières nous ont permis de localiser quelques parties coûteuses dans le LINK.

Une connaissance approfondie du fonctionnement du LINK nous a été indispensable pour mener à bien ce travail.

Les mesures réalisées démontrent qu'une proportion non négligeable du temps CPU consommé par le LINK est à imputer à la lecture des fichiers d'entrée (fichiers objets des utilisateurs et fichiers-librairies).

Elles démontrent aussi que ce n'est pas le LINK en lui même qui est coûteux! C'est plutôt le fait que celui-ci n'est pas adapté au système d'exploitation TOPS-20. En effet, le LINK utilise les appels-systèmes TOPS-10. L'interprétation de ceux-ci, par l'interface PA1050, nécessite la majeure partie du temps CPU consommé durant l'édition de liens d'un programme.

Cette adaptation est une entreprise trop importante pour être réalisée dans le cadre d'un mémoire. Néanmoins, par comparaison des deux méthodes (TOPS-10 et TOPS-20) nous avons pu déterminer qu'il s'agit effectivement d'une amélioration.

De plus, nous avons montré qu'une meilleure organisation des librairies-systèmes peut, dans certains cas, apporter un gain appréciable en temps CPU et en lectures sur disque. Les mesures nécessaires pour pouvoir réaliser cette réorganisation (détermination de "noyaux" de modules dans les librairies-systèmes) pourraient faire l'objet d'une étude future.

De même, une analyse plus approfondie de certains programmes FORTRAN permettrait d'en identifier les particularités faisant qu'ils nécessitent beaucoup de temps CPU pour être reliés et chargés. Ceci pourrait amener à proposer d'autres améliorations au LINK.

BIBLIOGRAPHIE.

1. Relieurs et chargeurs.

[DB] BARRON D. W.
"Assemblers and Loaders."
American Elsevier, New York, 1969

[PW] PRESSER L., WHITHE J. R.
"Linkers and Loaders."
Computing Surveys, vol. 4, n. 3, Sept. 1972, p. 149

[JD] DONOVAN J. J.
"System Programming."
McGraw-Hill, International Student Edition, 1972

[PL] LEMAIRE P.
"Rapport Détaillé et Provisoire sur le LINK du DEC-20."
Documentation Interne du Service d'Informatique de
l'Institut Montéfiore, Université de Liège, 1979

[LRM] DIGITAL EQUIPMENT CORPORATION
"LINK Reference Manual."
Digital Software Notebooks, 1980

2. Mesures de performances.

[DK] KNUTH D. E.
"An Empirical Study of FORTRAN Programs."
Software - Practice and Experience,
vol. 2, 1971, pp. 105-133

[WW] WAITE W. M.
"A Sampling Monitor for Applications Programs."
Software - Practice and Experience,
vol. 3, 1973, pp. 75-79

[JC] COLLINS J. P.
"Performance Improvement of the CP-V Loader through use of the ADAM Hardware Monitor."
Performance Evaluation Review,
vol. 5, n. 2, Apr. 1976

[DF] FERRARI D.
"Computer System Performance Evaluation."
Prentice Hall, 1978

[JR] RAMAEKERS J.
"La Gestion des Logiciels."
Publications de l'Institut d'Informatique, F.U.N, 1979

[VF] FOSTER V. S.
"Performance Measurement of a PASCAL Compiler."
SIGPLAN Notices, vol. 15, n. 6, Jun. 1980

[JH] HUGHES J. H.
"DIAMOND - A Digital Analizer and Monitoring Device."
Performance Evaluation Review, vol. 9, n. 2, 1980

[PHD] DE RIVET P.
"Introduction à la Métrologie des systèmes."
Notes de cours, 1980

[BCM] BABONNEAU J-Y., CARPENTIER M., MORISSET G.
"Optimisation de la Gestion sur Disque des Bibliothèques de Programmes."
Rapport de travail, INRIA, Sept. 1980

3. Divers.

[MD] DEBAR M. E.
"DDT and Consorts."
Documentation Interne du Centre de Calcul, F.U.N., 1980

[MRM] DIGITAL EQUIPMENT CORPORATION
"MACRO Reference Manual."
Digital Software Notebooks, 1980

[MC20] DIGITAL EQUIPMENT CORPORATION
"TOPS-20 Monitor Calls Reference Manual."
Digital Software Notebooks, 1980

[MC10] DIGITAL EQUIPMENT CORPORATION
"TOPS-10 Monitor Calls Reference Manual."
Digital Software Notebooks, 1980

[MUG] DIGITAL EQUIPMENT CORPORATION
"MAKLIB Users Manual."
Digital Software Notebooks, 1980

[MO] DIGITAL EQUIPMENT CORPORATION
"DECsystem-20 Monitor Overview."
Documentation Interne, 1980

ANNEXES.

TABLE DES MATIERES.

1. EXEMPLES DE REL BLOCKS.

Nous présentons ici quelques exemples des REL BLOCKS les plus courants. Pour ce faire, nous avons utilisé un petit programme FORTRAN.

Outre le fait que le programme principal (SYNTHE) appelle diverses routines de la librairie FORTRAN (FORLIB), il fait également appel à la routine SUB001 de la librairie utilisateur LIBRAR.REL.

Le code source est suivit du résultat de la compilation tel qu'il peut être trouvé dans un fichier .LST produit par le compilateur FORTRAN.

Ensuite, un "DUMP" du fichier .REL donne le détails des différents REL BLOCKS générés. De cette manière, on peut remarquer que le fichier SYNTHE.REL est formé de 13 blocs des types 4,6,3,1,2,10,7 et 5.

(Remarquons encore qu'une valeur relogeable est signalée par un "'".)

SOURCE CODE

```
00001            PROGRAM SYNTHE
00002            INTEGER I
00003            I=0
00004            WRITE(5,1000)
00005   1000     FORMAT(1X,'MAIN BEGIN')
00006            I=I+1
00007            CALL SUB001(I)
00008            I=I+1
00009            CALL SUB001(I)
00010            I=I+1
00011            CALL SUB001(I)
00012            WRITE(5,1001)
00013   1001     FORMAT(1X,'MAIN END')
00014            END
```

OBJECT CODE

```
LINE    LOC     LABEL   GENERATED CODE

        0               JFCL    0,0
        1               JSP     16,RESET.
        2                       0,0
3       3               SETZB   2,I
4       4               MOVEI   16,2M
        5               PUSHJ   17,OUT.
        6               PUSHJ   17,FIN.
6       7               AOS     2,I
7       10              MOVEI   16,3M
        11              PUSHJ   17,SUB001
8       12              AOS     2,I
9       13              MOVEI   16,4M
        14              PUSHJ   17,SUB001
10      15              AOS     2,I
11      16              MOVEI   16,5M
        17              PUSHJ   17,SUB001
12      20              MOVEI   16,6M
        21              PUSHJ   17,OUT.
        22              PUSHJ   17,FIN.
```

```
14      23              MOVEI   16,1M
        24              PUSHJ   17,EXIT.

 ARGUMENT BLOCKS:

        25              0,,0
        26      1M:     0,,0
        27              777777,,0
        30      5M:     100,,I
        31              777777,,0
        32      4M:     100,,I
        33              777777,,0
        34      3M:     100,,I
        35              777773,,0
        36      2M:     0,,5
        37              0,,0
        40              0,,0
        41              340,,1000P
        42              0,,4
        43              777773,,0
        44      6M:     0,,5
        45              0,,0
        46              0,,0
        47              340,,1001P
        50              0,,3

FORMAT STATEMENTS (IN LOW SEGMENT):

5       2       1000P:  (1X,'
        3               MAIN
        4               BEGIN
        5               ')
13      6       1001P:  (1X,'
        7               MAIN
        10              END')


Dump of file DSK:SYNTHE.REL[4,457]

Record 1.       Type 4  (Entry)         Word count 1

        SYNTHE

Record 2.       Type 6  (Name)          Word count 2

        Program name:   SYNTHE
        Processor:      2       (VI10)
        Translator:     10      (FORTRAN)
        COMMON length:  000000

Record 3.       Type 3  (HiSeg)         Word count 2

        High segment origin:    400000'
        High segment break:     400000
        Low segment origin:     000000
        Low segment break:      000011

Record 4.       Type 1  (Code)          Word count 10

        000002' /       241433 026116       "(1X,'"
        000003' /       466031 147100       "MAIN "
        000004' /       412130 744634       "BEGIN"
```

```
        000005' /       235220 000000       "')   "
        000006' /       241433 026116       "(1X,'"
        000007' /       466031 147100       "MAIN "
        000010' /       426350 423522       "END')"

Record 5.       Type 1  (Code)          Word count 22

        400000' /       255000 000000   JFCL    0
        400001' /       265700 000000   JSP     16,0
        400002' /       000000 000000
        400003' /       403100 000001'  SETZB   2,1
        400004' /       201700 000000   MOVEI   16,0
        400005' /       260740 000000   PUSHJ   17,0
        400006' /       260740 000000   PUSHJ   17,0
        400007' /       350100 000001'  AOS     2,1
        400010' /       201700 000000   MOVEI   16,0
        400011' /       260740 000000   PUSHJ   17,0
        400012' /       350100 000001'  AOS     2,1
        400013' /       201700 000000   MOVEI   16,0
        400014' /       260740 000000   PUSHJ   17,0
        400015' /       350100 000001'  AOS     2,1
        400016' /       201700 000000   MOVEI   16,0
        400017' /       260740 000000   PUSHJ   17,0
        400020' /       201700 000000   MOVEI   16,0

Record 6.       Type 1  (Code)          Word count 22

        400021' /       260740 000000   PUSHJ   17,0
        400022' /       260740 000000   PUSHJ   17,0
        400023' /       201700 000000   MOVEI   16,0
        400024' /       260740 000000   PUSHJ   17,0
        400025' /       000000 000000
        400026' /       000000 000000
        400027' /       777777 000000
        400030' /       000100 000001'
        400031' /       777777 000000
        400032' /       000100 000001'
        400033' /       777777 000000
        400034' /       000100 000001'
        400035' /       777773 000000
        400036' /       000000 000005
        400037' /       000000 000000
        400040' /       000000 000000
        400041' /       000340 000002'

Record 7.       Type 1  (Code)          Word count 10

        400042' /       000000 000004
        400043' /       777773 000000
        400044' /       000000 000005
        400045' /       000000 000000
        400046' /       000000 000000
        400047' /       000340 000006'
        400050' /       000000 000003

Record 8.       Type 2  (Symbols)       Word count 22

RESET.  400001' Global request:  RH chained relocation
OUT.    400005' Global request:  RH chained relocation
FIN.    400006' Global request:  RH chained relocation
SUB001  400011' Global request:  RH chained relocation
SUB001  400014' Global request:  RH chained relocation
SUB001  400017' Global request:  RH chained relocation
```

```
OUT.    400021' Global request:  RH chained relocation
FIN.    400022' Global request:  RH chained relocation
EXIT.   400024' Global request:  RH chained relocation

Record 9.       Type 2  (Symbols)       Word count 22

I       000001' Local symbol
.VEND   000002' Local symbol
6M      400044' Local symbol
5M      400030' Local symbol
4M      400032' Local symbol
3M      400034' Local symbol
2M      400036' Local symbol
1M      400026' Local symbol
1001P   000006' Local symbol

Record 10.      Type 2  (Symbols)       Word count 4

1000P   000002' Local symbol
MAIN.   400000' Global definition

Record 11.      Type 10 (Internal Request)  Word count 6

        400023'400026'
        400016'400030'
        400013'400032'
        400010'400034'
        400004'400036'
        400020'400044'

Record 12.      Type 7  (Start)         Word count 1

        Start address:  400000'

Record 13.      Type 5  (End)           Word count 2

        High segment break:     400051'
        Low segment break:      000011'


SOURCE CODE


00001           SUBROUTINE SUB001(INDEX)
00002           WRITE(5,1000),INDEX
00003   1000    FORMAT(1X,'SUBROUTINE NUMBER 001/INDEX IS ',I10)
00004           RETURN
00005           END

OBJECT CODE

LINE    LOC     LABEL   GENERATED CODE

        0               636542,,202021

        SUB001:
1       0               MOVEM   16,.A0016
        1               MOVE    0,@0(16)
        2               MOVEM   0,INDEX
2       3       3M:
                        MOVEI   16,4M
        4               PUSHJ   17,OUT.
```

```
        5                MOVEI   16,5M
        6                PUSHJ   17,IOLST.
5       7        2M:
                         MOVE    16,.A0016
        10               POPJ    17,0

ARGUMENT BLOCKS:

        11               777773,,0
        12       4M:     0,,5
        13               0,,0
        14               0,,0
        15               340,,1000P
        16               0,,11
        17               0,,0
        20       5M:     1100,,INDEX
        21               4000,,0

FORMAT STATEMENTS (IN LOW SEGMENT):

3       2        1000P:  (1X,'
        3                SUBRO
        4                UTINE
        5                 NUMB
        6                ER 00
        7                1 / I
        10               NDEX
        11               IS ',
        12               I10)



Dump of file DSK:LIBRAR.REL[4,457]

Record 1.       Type 14 (Index)          Word count 177

        Module containing 1. entry point
         begins at block number 2 word number 0
                SUB001
        Module containing 1. entry point
         begins at block number 2 word number 130
                SUB002
        Module containing 1. entry point
         begins at block number 3 word number 60
                SUB003
        Module containing 1. entry point
         begins at block number 4 word number 10
                SUB004
        Next index block number 777777

Record 2.       Type 4  (Entry)          Word count 1

        SUB001

Record 3.       Type 6  (Name)           Word count 2

        Program name:   SUB001
        Processor:      2       (KI10)
        Translator:     10      (FORTRAN)
        COMMON length:  000000

Record 4.       Type 1  (Code)           Word count 2
```

```
        000014' /       000000 000005

Record 5.       Type 3  (HiSeg)          Word count 2

        High segment origin:    400000'
        High segment break:     400000
        Low segment origin:     000000
        Low segment break:      000015

Record 6.       Type 1  (Code)           Word count 12

        000002' /       241433 026116    "(1X,'"
        000003' /       516530 251236    "SUBRO"
        000004' /       526511 147212    "UTINE"
        000005' /       202352 546604    " NUMB"
        000006' /       426444 030140    "ER 00"
        000007' /       305005 720222    "1 / I"
        000010' /       472110 554100    "NDEX "
        000011' /       446464 023530    "IS ',"
        000012' /       445426 024400    "I10)'"

Record 7.       Type 1  (Code)           Word count 22

        400000' /       636542 202021    'SUB001'
        400001' /       202700 000013'   MOVEM   16,13
        400002' /       200036 000000    MOVE    @0(16)
        400003' /       202000 000001'   MOVEM   1
        400004' /       201700 000000    MOVEI   16,0
        400005' /       260740 000000    PUSHJ   17,0
        400006' /       201700 000000    MOVEI   16,0
        400007' /       260740 000000    PUSHJ   17,0
        400010' /       200700 000013'   MOVE    16,13
        400011' /       263740 000000    POPJ    17,0
        400012' /       777773 000000
        400013' /       000000 000005
        400014' /       000000 000000
        400015' /       000000 000000
        400016' /       000340 000002'
        400017' /       000000 000011
        400020' /       000000 000000

Record 8.       Type 1  (Code)           Word count 3

        400021' /       001100 000001'
        400022' /       004000 000000

Record 9.       Type 2  (Symbols)        Word count 22

SUB001  400001' Global definition
OUT.    400005' Global request: RH chained relocation
IOLST.  400007' Global request: RH chained relocation
INDEX   000001' Local symbol
.A0016  000013' Local symbol
.VEND   000002' Local symbol
5M      400021' Local symbol
4M      400013' Local symbol
3M      400004' Local symbol

Record 10.      Type 2  (Symbols)        Word count 6

2M      400010' Local symbol
1M      000001' Local symbol
```

```
1000P   000002' Local symbol

Record 11.      Type 10 (Internal Request) Word count 2

        400004'400013'
        400006'400021'

Record 12.      Type 5  (End)           Word count 2

        High segment break:     400023'
        Low segment break:      000015'
```

2. LA COLLECTE DES INFORMATIONS.

2.1. Préliminaires.

Les routines de collecte de données ont été introduites dans le programme exécutable LINK à l'aide de DDT (l'outil d'aide à la mise au point de programmes). Cet utilitaire permet de suivre l'exécution d'un programme, de l'interrompre à certaines adresses ("BREAK POINTS"), de vérifier le contenu des accumulateurs et des variables et, surtout, de modifier dynamiquement le code ou les données (cf. [MD]).

C'est cette dernière possibilité que nous avons utilisé. En effet, tout programme exécutable contient une zone de 100h mots appelée "PATCH AREA" (adresses PAT.. jusque PAT..+77). Celle-ci peut recevoir des ajoutes au code. Certaines instructions du LINK seront donc remplacées par des appels aux routines d'acquisition placées dans la "PATCH AREA". Lorsque la collecte des données est terminée, l'exécution du LINK reprend sont cours normal à l'endroit où elle a été interrompue.

Les avantages de cette manière de procéder sont le coût réduit et la facilité de la mise au point. Du fait que l'on travaille sur un programme exécutable, il n'y a pas besoin de compiler et relier les différents modules du LINK!

2.2. Les routines d'acquisition.

Nous avons utilisé trois routines d'acquisition de données.

La première prend le temps CPU au début de l'exécution du LINK. L'utilisation de l'appel-système RUNTM fournit le temps CPU consommé par le processus entre le moment de sa création et le moment de l'appel. Néanmoins, il faut tenir compte du fait que c'est le même processus qui peut servir à la compilation, à l'édition de liens-chargement et, finalement, à l'exécution du programme de l'utilisateur! dés lors, mesurer le temps à la fin du LINK ne donnera pas toujours le résultat voulu. Il faut nécessairement mesurer le temps CPU consommé au début et à la fin du LINK et, ensuite, faire la différence des deux valeurs obtenues.

La seconde calcule la taille du programme exécutable construit par le LINK ("CORE IMAGE") et la place dans un mot bien déterminé de la "PATCH AREA".

La troisième rassemble toutes les informations et crée un enregistrement qui est, ensuite, copié dans le fichier PS:[2]INF.M-ADAMS[3]LNKPRF.DTA. La protection de ce fichier est telle

- qu'un seul utilisateur à la fois peut y accéder,
- que le seul accès autorisé est l'accès "APPEND".

De cette façon, nous avons pu éviter les conflits d'accès simultanés et assurer la sécurité des données déjà acquises.

2.2.1. Le calcul de la taille d'un programme.

L'espace mémoire (virtuel) d'un processus est considéré comme étant formé de deux segments:

- le bas segment ("LOW SEGMENT") occupant les pages 0b à 377b de l'espace d'adressage,

- le haut segment ("HIGH SEGMENT") occupant les pages 400b à 777b

(Cette division doit permettre la génération aisée de programmes réentrants. Le code peut être placé dans le haut segment et les données dans le bas segement. Cependant, les compilateurs ne tiennent pas toujours compte de cette possibilite!)

Pour calculer la taille d'un programme exécutable il faut donc faire la somme des zones occupées dans les deux segments. C'est ce que réalise la routine GETSIZ située en PAT..+56 jusque PAT..+74. Elle utilise les adresses de debut et de fin des zones occupées dans les deux segments (ces adresses sont disponibles dans le module LNKLOW du LINK). A partir du nombre se mots, GETSIZ calcule ne nombre de pages de 512 mots occupées par le programme. (C'est cette méthode que le LINK utilise pour calculer la taille du programme lorsque l'utilisateur a demandé un fichier .MAP.)

Cette routine est appelée au moment ou se décide si oui ou non il faut créer un fichier .MAP (i.e. en LODXIT+15). Ceci est nécessaire pour assurer la cohérence avec les informations fournies dans le fichier .MAP. En effet, il peut arriver que les variables utilisées soient modifiées dans la suite du programme.

Précisons, de plus, que le calcul de la taille du "CORE IMAGE" devient beaucoup plus complexe dans le cas où interviennent des "OVERLAYS". Dans ce cas, la routine GETSIZ ne fournira pas un résultat correct. Cette technique étant très rarement utilisée (2 ou 3 personnes sur l'ensemble des utilisateurs), on peut se permettre de ne pas en tenir compte.

2.2.2. La routine principale.

La routine principale (COLECT, adresses PAT..+5 à PAT..+54) rassemble les données manquantes (le nombre de symboles globaux et le langage source) et calcule le temps CPU consommé depuis le début de l'exécution du LINK.

De plus, elle construit un enregistrement contenant ces informations et le copie dans le fichier.

Cette routine est appelée au moment où se décide si oui ou non il y a lieu de créer un fichier .EXE. Le temps CPU mesuré est donc un temps par défaut. Ceci est nécessaire du fait que le LINK se détruit lui-même à la fin de son exécution. Les informations pourraient être perdues si l'on ne prenait pas cette précaution.

2.2.3. L'enregistrement.

L'enregistrement prend deux mots de 36 bits. Son format est le suivant:

	0	1 — 17	18 — 35
PREMIER MOT:	0	TAILLE	TEMPS CPU
SECOND MOT:	1	LANGAGE	# SYMBOLES

Le premier bit de chaque mot permet de différencier les deux mots de l'enregistrement. Ceci est utile dans le cas d'une perte éventuelle d'un des deux mots (il n'y a pas de danger d'altérer les informations étant donné que la taille n'excède pas 512 pages (9 bits) et que le langage source du programme principal est codé sous la forme d'un index dans la table des langages connus du LINK (valeurs de 0 à 20)).

2.2.4. Le code.

1. Appel à la routine GETIME:

```
link+4  /       move    wc,0

devient:

link+4  /       jrst    pat..
```

2. Appel à la routine GETSIZ:

```
lodxit+15/      move    1,mapsw

devient:

lodxit+15/      jrst    pat..+56
```

3. Appel à la routine COLECT:

```
savtst  /       skipe   io.ptr+16

devient:

savtst  /       jrst    pat..+5

;
;ROUTINE GETIME
;
pat..   /       movei   1,400000 ; .fbslf in ac 1
pat..+1 /       jsys    15      ;RUNTM jsys
pat..+2 /       movem   1,pat..  ;save it for later
pat..+3 /       move    wc,0
pat..+4 /       jrst            ;go back
;
;ROUTINE COLECT
;
pat..+5 /       movei   1,400000 ;.fbslf in ac 1
pat..+6 /       jsys    15      ;RUNTM jsys
```

```
pat..+7 /       sub     1,pat..  ;substract previous
pat..+10/       hrrm    1,pat..+1 ;save runtime
pat..+11/       hlrz    1,mntype  ;get main compiler type
pat..+12/       hrlzm   1,pat..+2 ;save it
pat..+13/       move    1,gsym    ;get number of globals
pat..+14/       hrrm    1,pat..+2 ;save it
pat..+15/       hrlzi   1,100001  ;[gj%old+gj%sht]
pat..+16/       hrroi   2,pat..+47 ;get file specs
pat..+17/       jsys    20        ;GTJFN jsys
pat..+20/       erjmp   pat..+44  ;in case of error
pat..+21/       movem   1,pat..+3 ;save jfn
pat..+22/       movei   2,20100   ;[of%app+of%rtd]
pat..+23/       jsys    21        ;OPENF jsys
pat..+24/       erjmp   pat..+41  ;in case of error
pat..+25/       move    2,pat..+1 ;write first word
pat..+26/       tlz     2,400000  ; of record
pat..+27/       jsys    51        ;BOUT jsys
pat..+30/       erjmp   pat..+41  ;in case of error
pat..+31/       move    2,pat..+2 ;write second word
pat..+32/       tlo     2,400000  ; of record
pat..+33/       jsys    51        ;BOUT jsys
pat..+34/       jfcl
pat..+35/       move    1,pat..+3 ;get jfn back
pat..+36/       jsys    22        ;CLOSE jsys
pat..+37/       jfcl              ;in case of error
pat..+40/       jrst    pat..+44  ;ok, go back
pat..+41/       move    1,pat..+3 ;get jfn back
pat..+42/       jsys    23        ;RLJFN jsys
pat..+43/       jfcl
pat..+44/       skipe   io.ptr+16 ;return to sender
pat..+45/       jrst    jbsave
pat..+46/       jrst    savtst+2
pat..+47/       ps:<i
pat..+50/       nf.m-
pat..+51/       adans
pat..+52/       >lnkp             ;file specs
pat..+53/       rf.dt
pat..+54/       a
pat..+55/       0
;
;ROUTINE GETSIZ
;
pat..+56/       setz    r1,       ;for index
pat..+57/       move    1,hl.sl(r1)  ;get hl.sl
pat..+60/       add.    1,.pgsiz  ;round up to a page
pat..+61/       andcm.  1,.pgsiz  ; ...
pat..+62/       lsh     1,-11     ;number of pages
pat..+63/       addm    1,pat..+55 ;add to counter
pat..+63/       skipn   r1        ;high segment done?
pat..+64/       skipn   hl.s2     ;no, but is there any?
pat..+65/       skipa
pat..+66/       aoja    r1,pat..+57  ;go do it
pat..+67/       move    1,pat..+55   ;store total size
pat..+70        hrlzm   1,pat..+1    ;in first word
pat..+71/       move    1,mapsw      ;need a map?
pat..+72/       came    1,.erduz+12  ;  ...
pat..+73/       jrst    lnkxit       ;no
pat..+74/       jrst    lnkmap       ;yes
```

3. LES RESULTATS.

3.1. Les histogrammes.

Les histogrammes qui suivent décrivent les distributions des temps CPU consommés lors des éditions de liens-chargements des programmes, de la tailles des programmes exécutables produit et des nombres de symboles globaux de ces programmes.

Pour le premier histogramme (temps CPU consommés) nous avons choisi un intervalle de 100 millisecondes.

Pour le second histogramme (taille des programmes) nous avons utilisé l'unité c'est-à-dire la page de 512 mots.

Finalement, le troisième histogramme (nombre des symboles globaux) est construit sur base d'un intervalle de 5 symboles.

Dans chaque cas, une étoile représente dix observations.

Chaque histogramme est suivi de sa distribution cumulée correspondante.

PREMIER HISTOGRAMME: TEMPS CPU CONSOMMES.

(9919 observations)

```
 1!   103!**********
 2!   124!************
 3!   204!********************
 4!   934!*****************************************************************************************
 5!  1381!*************************************************************************************************************************
 6!  1065!*********************************************************************************************************
 7!   771!*************************************************************************
 8!   767!************************************************************************
 9!   526!*************************************************
10!   488!************************************************
11!   412!****************************************
12!   418!****************************************
13!   336!*********************************
14!   260!*************************
15!   204!********************
16!   171!*****************
17!   137!*************
18!   166!*****************
19!   171!*****************
20!   116!***********
21!    76!*******
22!    51!*****
23!    75!*******
24!    76!*******
25!    69!******
26!    51!*****
27!    55!*****
28!    52!*****
29!    62!******
30!    48!****
31!    48!****
32!    51!*****
33!    40!****
34!    29!**
35!    19!*
36!    18!*
37!    25!**
38!    17!*
39!    15!*
40!    13!*
41!    22!**
42!    20!**
43!    18!*
44!    18!*
```

distribution cumulée correspondant à l'histogramme précédent

values	occ's	20%	40%	60%	80%	100%
1	103					
2	227	*				
3	431	*				
4	1365	*				
5	2746	*				
6	3811	*				
7	4582	*				
8	5349	*				
9	5875	*				
10	6363	*				
11	6775	*				
12	7193	*				
13	7529	*				
14	7789	*				
15	7993	*				
16	8164	*				
17	8301	*				
18	8467	*				
19	8638	*				
20	8754	*				
21	8830	*				
22	8881	*				
23	8956	*				
24	9032	*				
25	9101	*				
26	9152	*				
27	9207	*				
28	9259	*				
29	9321	*				
30	9369	*				
31	9417	*				
32	9468	*				
33	9508	*				
34	9537	*				
35	9556	*				
36	9574	*				
37	9599	*				
38	9616	*				
39	9631	*				
40	9644	*				
41	9666	*				
42	9686	*				
43	9704	*				
44	9722	*				
45	9734	*				
46	9742	*				
47	9751	*				
48	9756	*				
49	9762	*				
50	9771	*				
51	9777	*				
52	9784	*				
53	9791	*				

```
  1!   508!**************************************************
  2!   503!**************************************************
  3!   307!*******************************
  4!   328!********************************
  5!   185!******************
  6!   505!**************************************************
  7!   680!********************************************************************************
  8!   593!***********************************************************
  9!   406!****************************************
 10!   179!******************
 11!   340!**********************************
 12!   114!***********
 13!   206!********************
 14!   165!****************
 15!   453!*********************************************
 16!   288!****************************
 17!   165!****************
 18!    50!*****
 19!    54!*****
 20!    76!*******
 21!   100!**********
 22!    88!********
 23!    79!*******                    DEUXIEME HISTOGRAMME:  CORE IMAGE SIZES.
 24!    57!*****
 25!   126!************
 26!   139!**************
 27!    96!*********                                        (9919 observations)
 28!   160!****************
 29!    66!******
 30!   194!*******************
 31!   253!*************************
 32!   100!**********
 33!    96!*********
 34!    52!*****
 35!    76!*******
 36!    55!*****
 37!    36!***
 38!    48!****
 39!    21!**
 40!    48!****
 41!    48!****
 42!    29!**
 43!    21!**
 44!    11!*
 45!    12!*
 46!     7!
 47!     2!
 48!     6!
 49!     4!
 50!    28!**
 51!     9!
 52!     3!
 53!     9!
 54!     7!
 55!     3!
 56!    11!*
 57!    13!*
 58!    16!*
 59!     5!
 60!    37!***
 61!    43!****
 62!    23!**
 63!     3!
```

distribution cumulée pour l'histogramme précédent

values	occ's	20%	40%	60%	80%	100%
1	514	*				
2	1017	*				
3	1324	*				
4	1652	*				
5	1837	*				
6	2342	*				
7	3022	*				
8	3615	*				
9	4021	*				
10	4200	*				
11	4540	*				
12	4654	*				
13	4860	*				
14	5025	*				
15	5478	*				
16	5766	*				
17	5931	*				
18	5981	*				
19	6035	*				
20	6111	*				
21	6211	*				
22	6299	*				
23	6378	*				
24	6435	*				
25	6561	*				
26	6700	*				
27	6796	*				
28	6956	*				
29	7022	*				
30	7216	*				
31	7469	*				
32	7569	*				
33	7665	*				
34	7717	*				
35	7793	*				
36	7848	*				
37	7884	*				
38	7932	*				
39	7953	*				
40	8001	*				
41	8049	*				
42	8078	*				
43	8099	*				
44	8110	*				
45	8122	*				
46	8129	*				
47	8131	*				
48	8137	*				
49	8141	*				
50	8169	*				
51	8178	*				
52	8181	*				
53	8190	*				
54	8197	*				
55	8200	*				
56	8211	*				
57	8224	*				
58	8240	*				
59	8245	*				
60	8282	*				
61	8325	*				
62	8348	*				

3.1.3. Les nombre de symboles globaux.

```
 35!   229!**********************
 40!    43!****
 45!   122!************
 50!    23!**
 55!    80!********
 60!    65!******
 65!   502!**************************************************
 70!   832!********************************************************************************
 75!   675!*************************************************************************
 80!   440!*******************************************
 85!   205!********************
 90!   208!********************
 95!    59!*****
100!   105!**********
105!    44!****
110!   162!****************
115!    37!***
120!   130!*************
125!    83!********
130!    43!****
135!    84!********
140!    48!****
145!    73!*******
150!    28!**
155!    16!*
160!    16!*
165!    41!****
170!    38!***
175!    29!**
180!    63!******
185!    28!**
190!    70!*******
195!   119!***********
200!     2!
205!    21!**
210!     7!
215!    30!***
220!    95!*********
225!   264!**************************
230!    14!*
235!    49!****
240!   228!**********************
245!    36!***
250!    36!***
255!    19!*
260!     2!
265!    16!*
270!    25!**
275!    36!***
280!    28!**
285!    44!****
290!     1!
300!     1!
310!     2!
315!     1!
320!  1498!**************************************************************************************
```

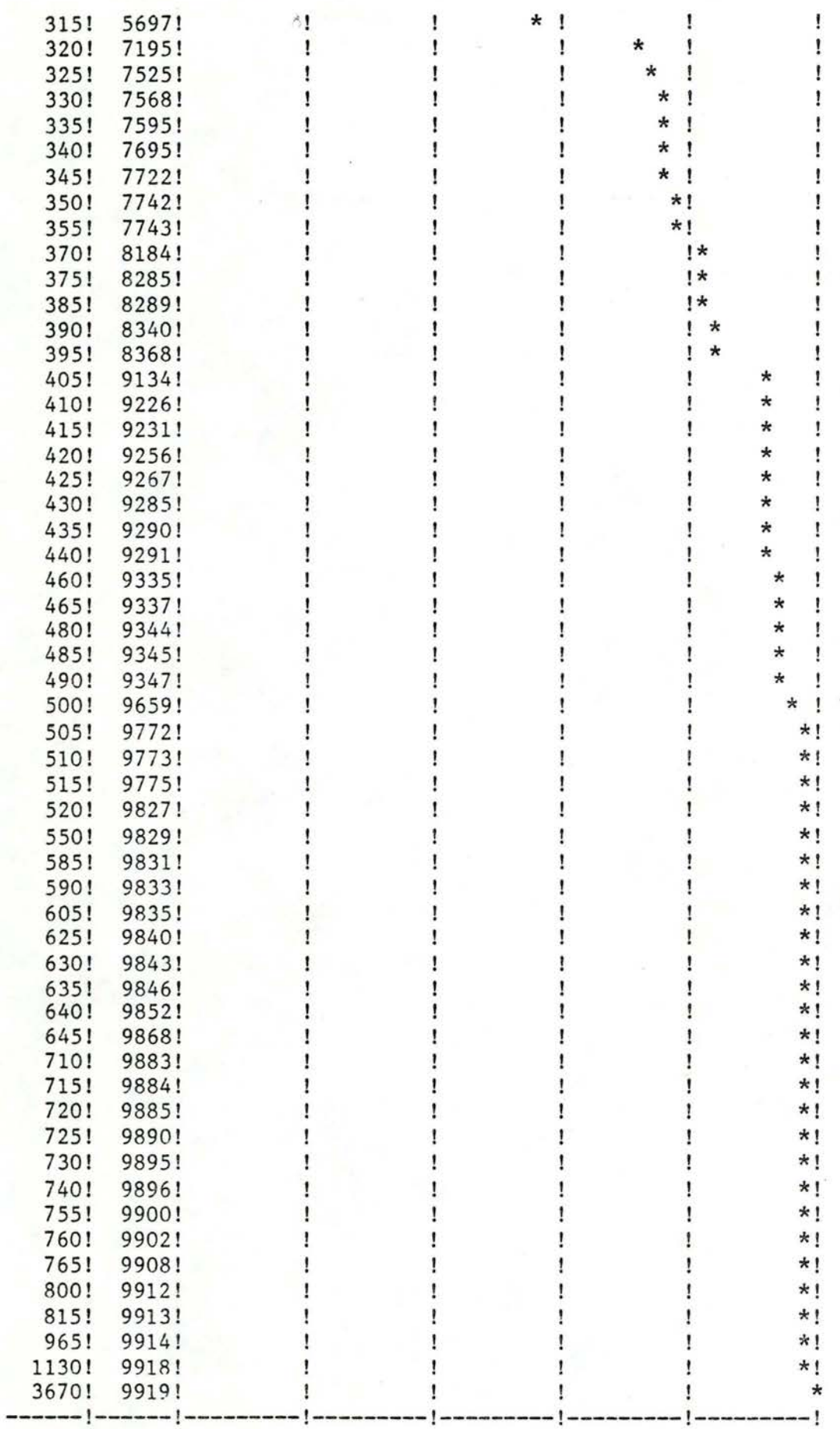
TROISIEME HISTOGRAMME: NOMBRE DE SYMBOLES GLOBAUX.

(9919 observations)

```
 325!   330!**********************************
 330!    43!****
 335!    27!**
 340!   100!**********
 345!    27!**
 350!    20!**
 355!     1!
 370!   441!*********************************************
 375!   101!**********
 385!     4!
 390!    51!*****
 395!    28!**
 405!   766!*************************************************************************************
 410!    92!*********
 415!     5!
 420!    25!**
 425!    11!*
 430!    18!*
 435!     5!
 440!     1!
 460!    44!****
 465!     2!
 480!     7!
 485!     1!
 490!     2!
 500!   312!********************************
 505!   113!***********
 510!     1!
 515!     2!
 520!    52!*****
 550!     2!
 585!     2!
 590!     2!
 605!     2!
 625!     5!
 630!     3!
 635!     3!
 640!     6!
 645!    16!*
 710!    15!*
 715!     1!
 720!     1!
 725!     5!
 730!     5!
 740!     1!
 755!     4!
 760!     2!
 765!     6!
 800!     4!
 815!     1!
 965!     1!
1130!     4!
3670!     1!
```

distribution cumulée pour l'histogramme précédent

values	occ's	20%	40%	60%	80%	100%
35	229	*				
40	272	*				
45	394	*				
50	417	*				
55	497	*				
60	562	*				
65	1064	*				
70	1896	*				
75	2571	*				
80	3011	*				
85	3216	*				
90	3424	*				
95	3483	*				
100	3588	*				
105	3632	*				
110	3794	*				
115	3831	*				
120	3961	*				
125	4044	*				
130	4087	*				
135	4171	*				
140	4219	*				
145	4292	*				
150	4320	*				
155	4336	*				
160	4352	*				
165	4393	*				
170	4431	*				
175	4460	*				
180	4523	*				
185	4551	*				
190	4621	*				
195	4740	*				
200	4742	*				
205	4763	*				
210	4770	*				
215	4800	*				
220	4895	*				
225	5159	*				
230	5173	*				
235	5222	*				
240	5450	*				
245	5486	*				
250	5522	*				
255	5541	*				
260	5543	*				
265	5559	*				
270	5584	*				
275	5620	*				
280	5648	*				
285	5692	*				
290	5693	*				
300	5694	*				
310	5696	*				

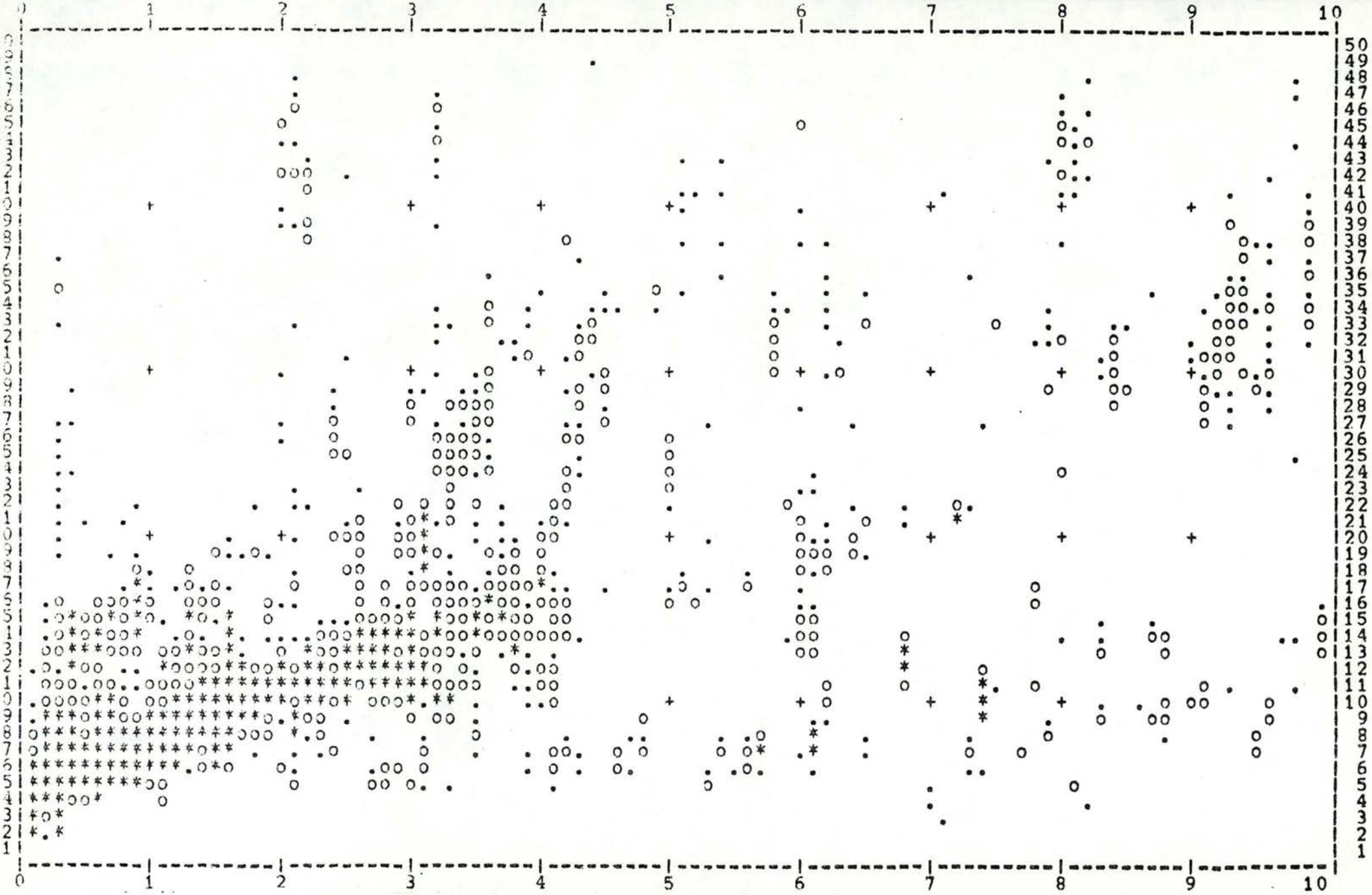


3.2. Les graphiques globaux.

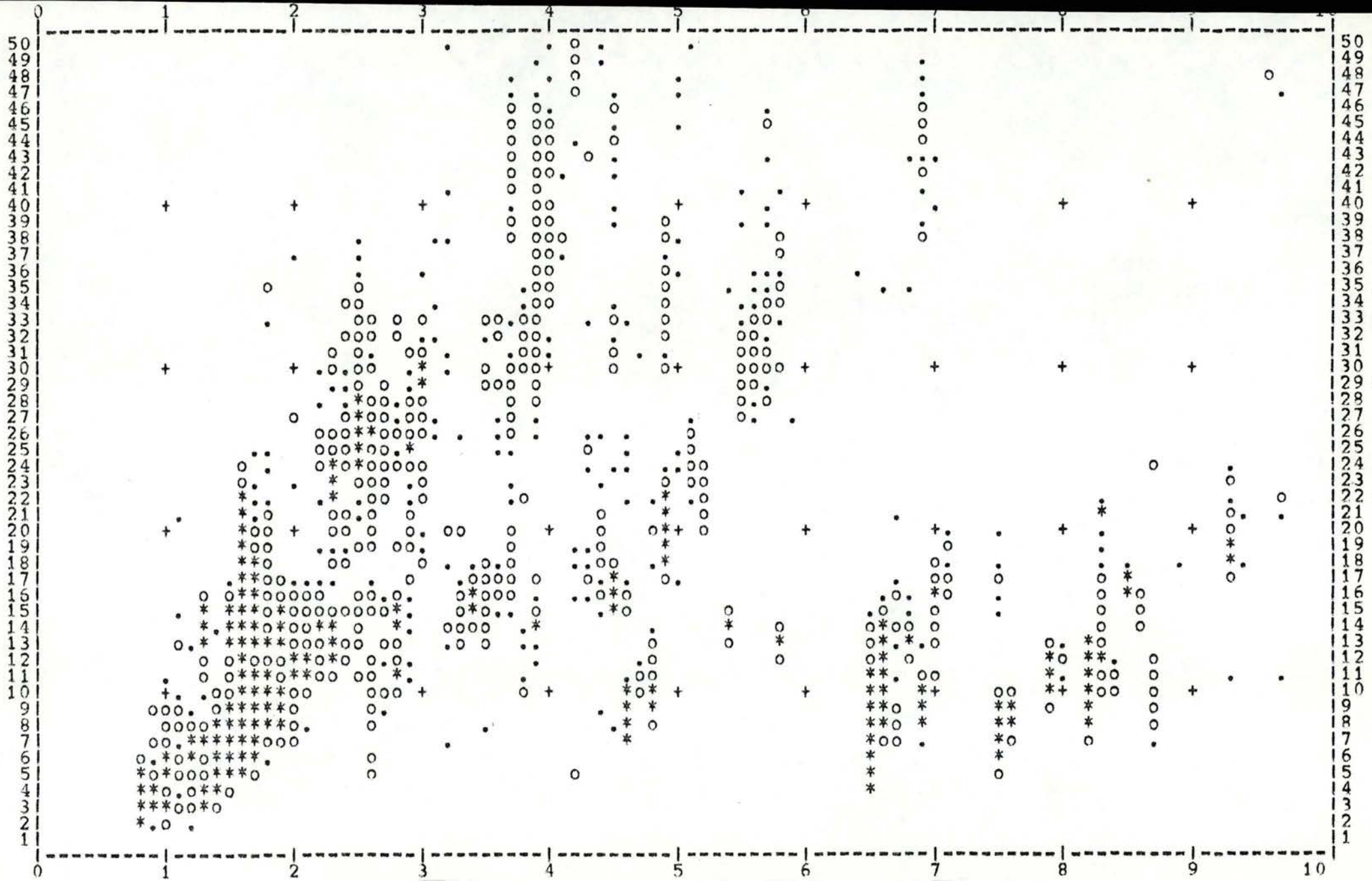
Les signes utilisés pour les graphiques sont les suivant:

- Un point signale l'occurrence d'une édition de lien-chargement d'un programme décrit par les coordonnées correspondantes,
- Un "o" signale un nombre d'occurrences supérieur à 1 mais inférieur à 10,
- Une étoile signale plus de 10 occurrences.

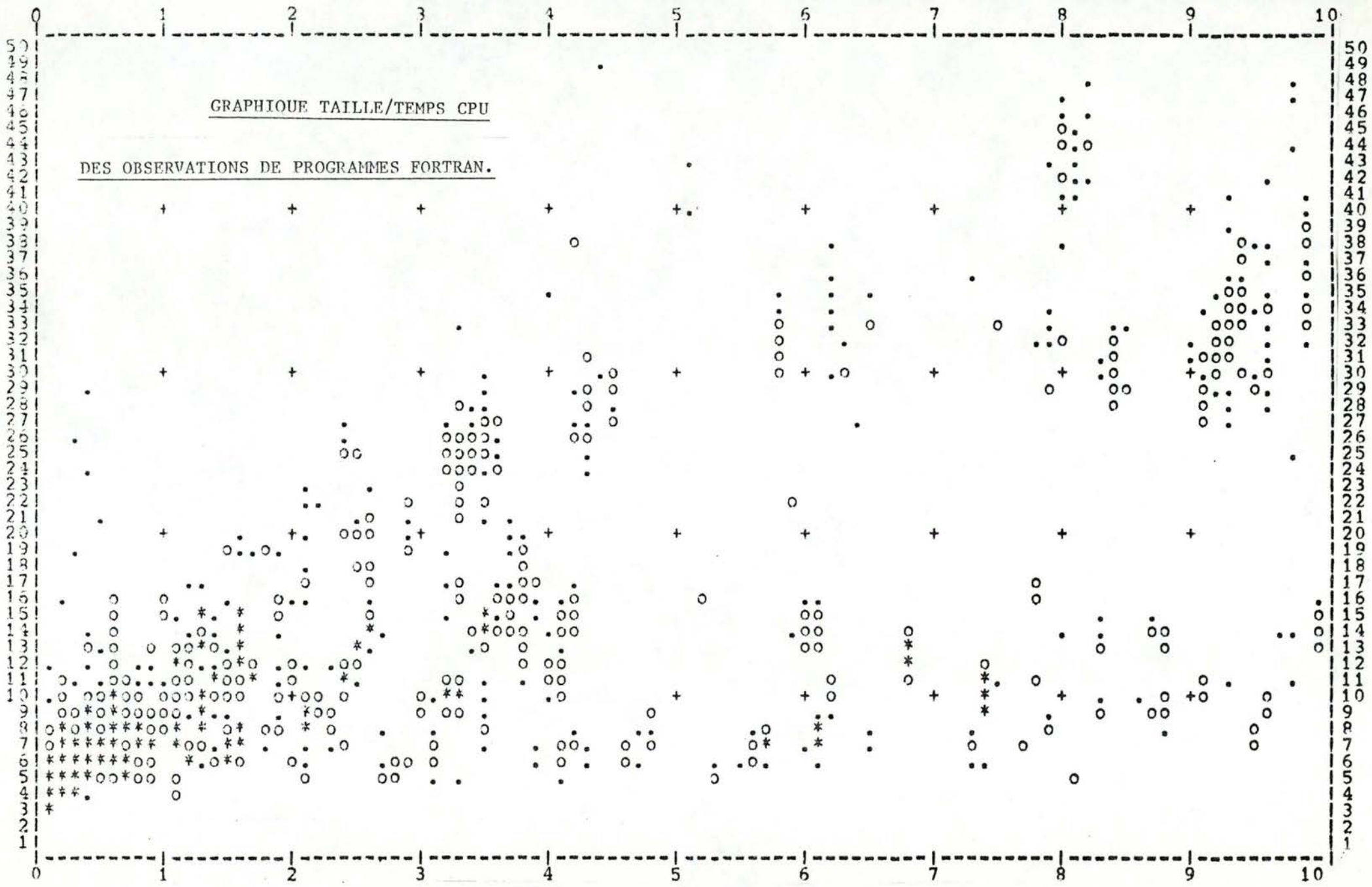
Les unités utilisées et les facteurs d'échelle sont donnés pour chaque graphique séparément.

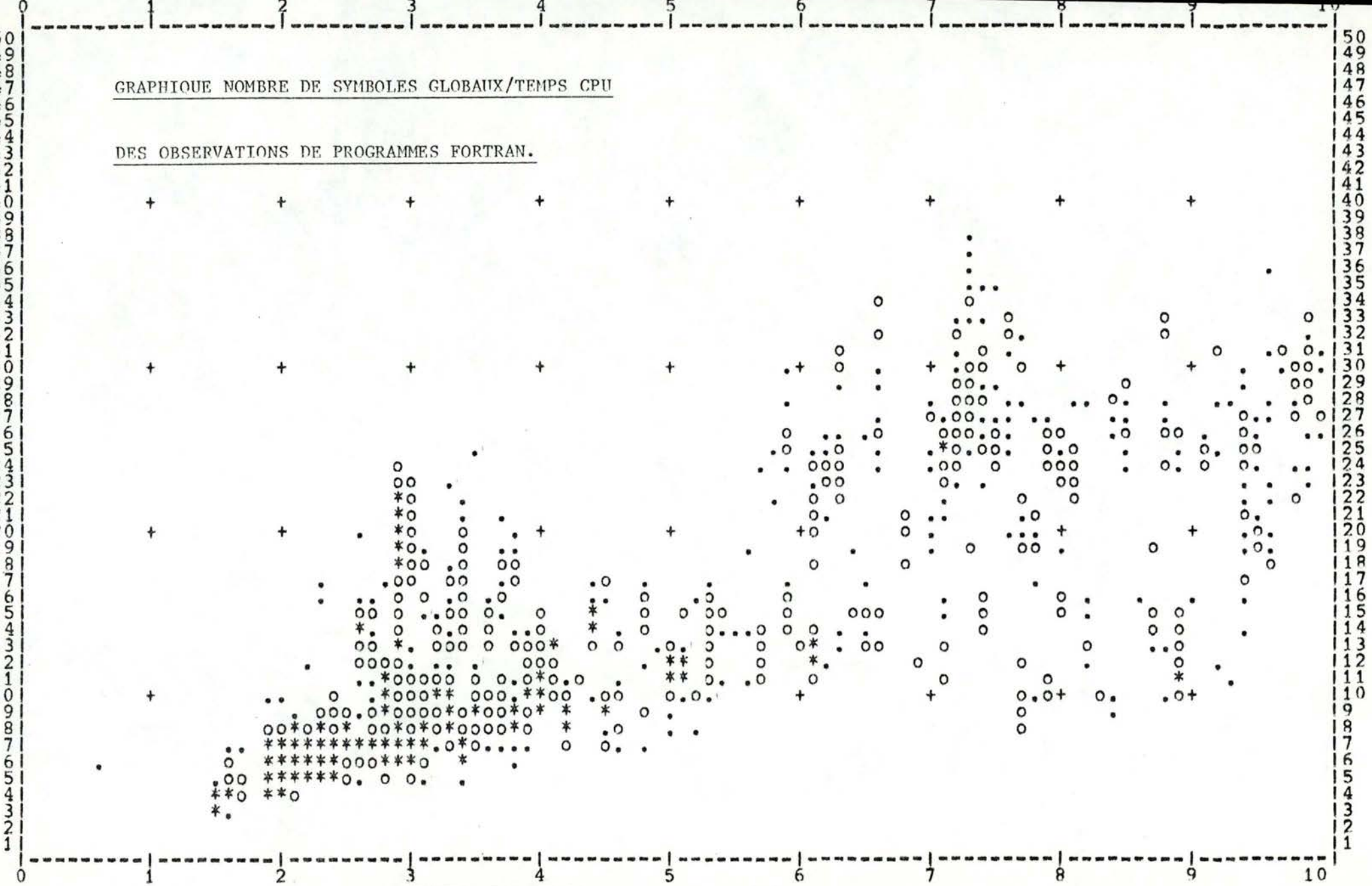


minimum x value: 0, scale factor for x axis: 1
minimum y value: 0, scale factor for y axis: 100
number of hits: 8743, frustrations: 1176

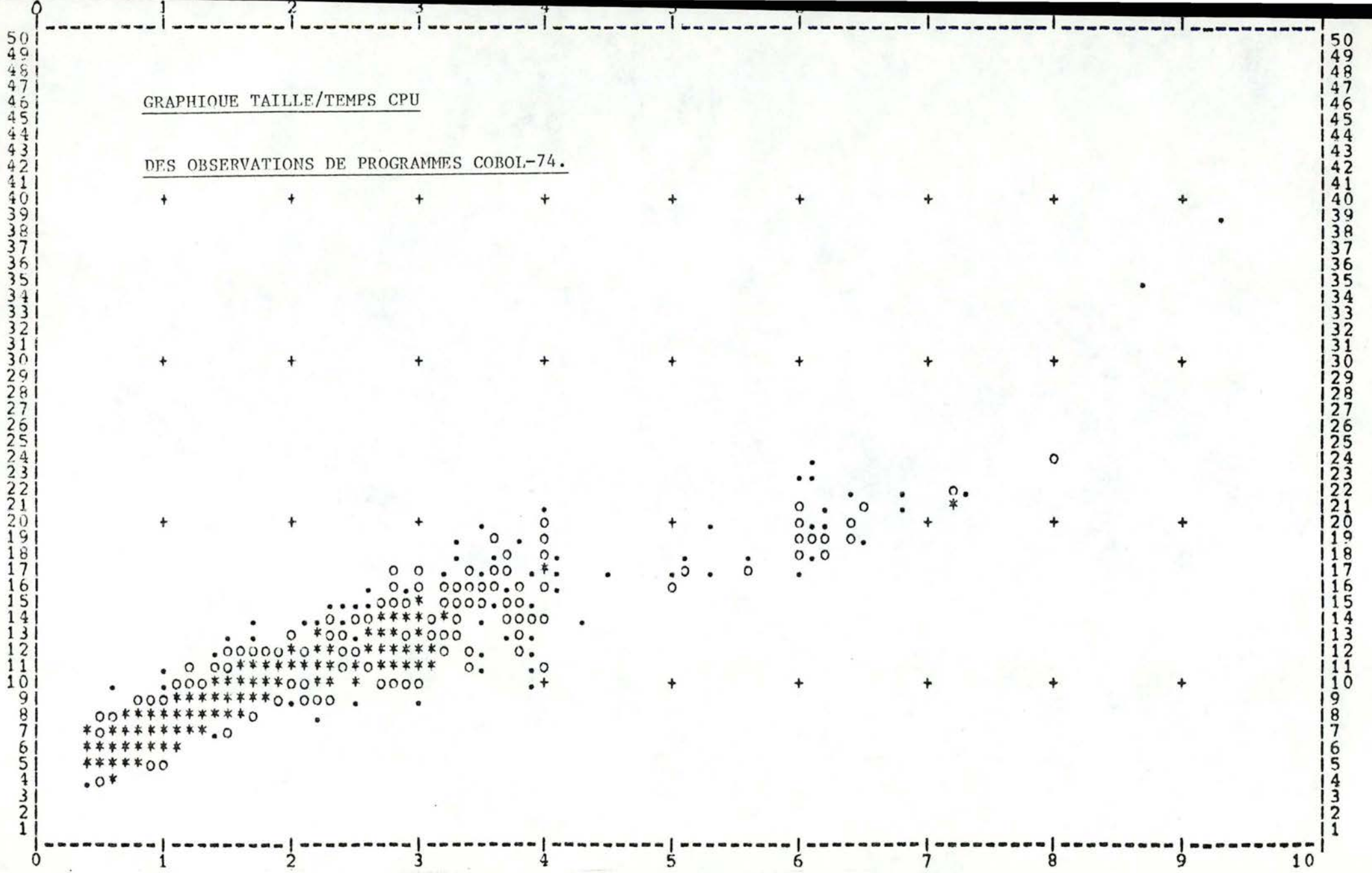


minimum x value: 0, scale factor for x axis: 5
minimum y value: 0, scale factor for y axis: 100
number of hits: 9203, frustrations: 716

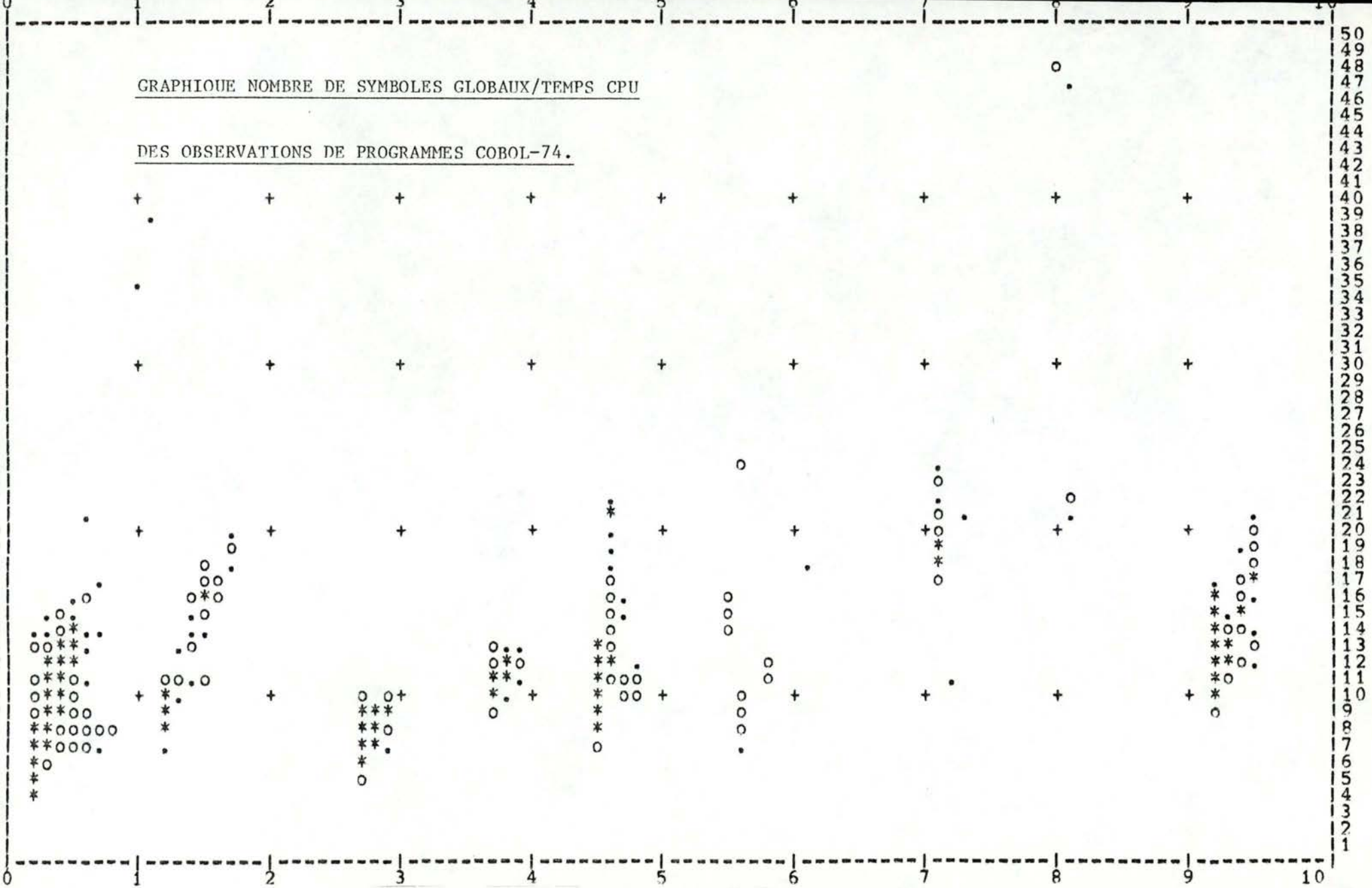

GRAPHIQUE TAILLE/TEMPS CPU
DES OBSERVATIONS DE PROGRAMMES FORTRAN.
0 1 2 3 4 5 6 7 8 9 10
minimun x value: 0, scale factor for x axis: 1
minimum y value: 0, scale factor for y axis: 100
number of hits: 3058, frustrations: 1151



minimum x value: 50, scale factor for x axis: 1
minimum y value: 0, scale factor for y axis: 100
number of hits: 3433, frustrations: 776



minimum x value: 0, scale factor for x axis: 1
minimum y value: 0, scale factor for y axis: 100
number of hits: 4031, frustrations: 3



minimum x value: 320, scale factor for x axis: 2
minimum y value: 0, scale factor for y axis: 100
number of hits: 3978, frustrations: 56

4. MYLOOK.

4.1. Le programme.

```
title mylook
search iomac,monsym
.require iomac
extern fatal,warn
sall

;
;conditional assembly switches
;
ftpal050==1
;
;ac's
;
count=5
obs=10
;
;parameters
;
tablen=100k     ;length of tables
cr=15           ;carriage return
ifn ftpal050,<monupc=papage+150>        ;return from PA1050
;                                       ;P-Count
;opdef's
;
opdef   call    [pushj  17,]
opdef   return  [popj   17,]
;
;
;
start:  reset
        move    17,[iowd  ^d20,stack]
        setz    count,

;
;Tell the world what we are doing
;
        outstr  (<
[MYLOOK Monitoring LINK]>)
;
;get the command string
;
inrel:  outstr  (<
command string:...>)
        instr   (^d80,cr,cmdlin)
        setz    1,
        idpb    1,2
;
;get number of times user wants to repeat the observation
;
inobs:  outstr  (<
how many observations:...>)
        inl0
         erjmp  [call  warn
                 jrst  inobs]
        movem   2,obs
        movem   2,mobs
;
;get the interval between looks
```

```
;
inint:  outstr  (<
interval between looks (in ms):...>)
        inl0
         erjmp  [call  warn
                 jrst  inint]
        movem   2,intrvl

;
;create inferior fork
;
crtfrk: move    1,[1b1]
        cfork
         erjmp  fatal
        movem   1,frkfrk
;
;if user wants, map PA1050 page into ourself
;
ifn ftpal050,<
mppage: hrlz    1,frkfrk
        hrri    1,733
        move    2,[xwd  .fhslf,500]
        move    3,[pm%rd]
        pmap
         erjmp  fatal
>;end ifn ftpal050
;
;get jfn
;
take:   move    1,[gj%old+gj%sht]
        hrroi   2,[asciz/sys:link.exe/]
        gtjfn
         erjmp  fatal
        movem   1,prgjfn
;
;tell'm how many still to do
;
        outstr  (<#>)
        move    2,obs
        out10
        outstr  (<
>)
;
;get program in it
;
gtprgr: hrlz    1,frkfrk
        hrr     1,prgjfn
        get
         erjmp  fatal
;
;initialize PSI system
;
psinit: movei   1,.fhslf
        move    2,[levtab,,chntab]
        sir
        movei   2,1b19  ;fork termination
        aic
        eir
;
;stick command string into input buffer
;
        movei   1,.priou
        move    3,[point  7,cmdlin]
```

```
stilp:  ildb    2,3
        jumpe   2,.+3
        sti
        jrst    stilp
;
;start inferior fork
;
rtfrk:  move    1,frkfrk
        setz    2,
        sfrkv
         erjmp  fatal
;
;get runtime
;
gtrunt: move    1,frkfrk
        runtm
        movem   1,time
;
;do the look
;
lkloop: move    1,intrvl
        disms
        move    1,frkfrk        ;get PC
        rfsts
        hlrz    1,1
        andi    1,7
        jumpn   1,notrun
        hrrz    2,2
ifn ftpal050,<
        caile   2,700000
         move   2,monupc
>;end ifn ftpal050
        soj     2,
        lsh     2,-3
        aos     frktab(2)
notrun: aos     status(1)
        aoja    count,lkloop
;
;here on inferior fork termination
;
frktrm: move    1,frkfrk
        runtm
        sub     1,time
        addm    1,mean
;
;clear PSI
;
        cis

;
;kill the fork
        move    1,frkfrk
        kfork
         erjmp  fatal
        sojg    obs,crtfrk
;
;it's done, now make a fine histogram
;
        outstr  (<
[RESULTS]>)
        outstr  (<
mean runtime in ms.:>)
        move    2,mean
```

```
        idiv    2,mobs
        outl0
        setz    7,
        movem   count,table
        outstr  (<
-------------------------------------------------------------->)
stsbeg: caile   7,10
         jrst   stsend
        outstr  (<
>)
        move    4,table(7)
        skipn   4
         aoja   7,stsbeg
        move    1,ststab(7)
        psout
        move    2,4
        outl0
        aoja    7,stsbeg
stsend: outstr  (<
-------------------------------------------------------------->)
        setz    6,
histbd: cail    6,tablen
         jrst   histfg
        move    4,frktab(6)
        skipe   4
         call   line
        aoja    6,histbd
histfg: outstr  (<
-------------------------------------------------------------->)
        outstr  (<
[DONE]>)
        haltf
         jrst   .-1
page
;
;
;
line:   outstr  (²
³)
        move    2,6
        lsh     2,3
        out8
        outstr  ( ! )
        move    2,4
        outl0
        outstr  ( ! )
        move    2,4
        idivi   2,^d10
        skipn   2
         return
        caig    2,^d100
         jrst   usel0
        idivi   2,^d10
        move    4,2
        outstr  (<          x100!>)
        skipa
usel0:  move    4,2
stars:
        movei   1,"*"
        pbout
        sojg    4,.-1
        return
;
```

```
page
;
;data
;
frkfrk: block   1
filjfn: block   1
prgjfn: block   1
intrvl: block   1
pcword: block   1
mean:   block   1
time:   block   1
mobs:   block   1
levtab: pcword
        exp     0,0
chntab: exp     0,0,0,0,0,0,0,0,0,0
        exp     0,0,0,0,0,0,0,0,0
        xwd     1,frktrm
page
cmdlin: block   ^d20
table:  block   1
status: block   10
stack:  block   ^d20
ststab: point   7,[asciz/TOTAL: /]
        point   7,[asciz/RUN:   /]
        point   7,[asciz/ioWAIT:/]
        point   7,[asciz/HALT:  /]
        point   7,[asciz/ERROR: /]
        point   7,[asciz/WAIT:  /]
        point   7,[asciz/SLEEP: /]
        point   7,[asciz/BRFACK:/]
        point   7,[asciz/ERROR: /]

        loc     500000
ifn ftpal050,<
papage: block   1000>
frktab: block   tablen
        reloc

        end     start
```

4.2. Les résultats.

4.2.1. Avec PA1050.

Les profils d'exécutions du LINK sont donnés pour divers programmes en entrée. Plus précisément, il s'agit de quelques programmes de notre échantillon. La numérotation est celle utilisée dans le tableau de la figure 6-1, p.40.

Programme numero 2 (FORTRAN)

```
[MYLOOK Monitoring LINK]
command string:...ptfixe,ptfsou/go
how many observations:...50
interval between looks:...40
[RESULTS]
mean runtime in ms.:  1196
-------------------------------------------------------------
TOTAL:   2258
RUN:     2138
ioWAIT:     4
SLEEP:    116
-------------------------------------------------------------
     0!    28!**
    20!    22!**
467260!    12!*
540000!     6!
540010!    87!********
540030!     4!
540130!     2!
540260!     1!
541620!     2!
542660!     6!
546330!    42!****
547010!     2!
547100!     6!
550130!     4!
553760!     2!
554040!     1!
554640!    35!***
555460!     4!
556130!     4!
557470!    10!*
560310!     1!
560550!     3!
560720!     1!
561300!     2!
561320!     1!
561520!    10!*
562430!     4!
563100!     4!
563130!     3!
563540!     1!
564530!     5!
564740!     4!
565510!     1!
565560!     1!
565570!     1!
565600!     2!
```

```
     0!     11!*
    20!     10!*
467260!      6!
540000!      1!
540010!     83!********
540030!      4!
541620!      3!
542660!      3!
546330!     51!*****
547010!      6!
547100!      7!
550130!     10!*
553760!      6!
554040!      6!
554640!     54!*****
554700!      2!
555260!      1!
555460!      6!
555470!      1!
556030!      2!
556040!      1!
556050!      1!
556130!      4!
557470!     16!*
560300!      1!
560320!      2!
560550!      4!
560610!      1!
560650!      1!
560660!      1!
560720!      2!
561210!      1!
561300!      5!
561310!      1!
561330!      2!
561340!      2!
561610!      1!
562430!      7!
562550!      2!
563100!      8!
563120!      1!
563330!      1!
564740!      9!
565020!      1!
565510!     11!*
565520!      1!
565530!      1!
565540!      1!
565550!      1!
565560!      9!
565600!      4!
565610!      2!
566660!      5!
567350!      1!
567470!      1!
567570!     10!*
570220!     40!****
572220!      2!
612470!      3!
621750!     74!*******
622110!     36!***
623310!     70!*******
623700!     29!**
```

```
565610!      1!
567570!      3!
570220!     35!***
572220!      1!
612470!      5!
621750!     54!*****
622110!     29!**
623310!     67!******
623700!     23!**
625640!     39!***
625770!    137!*************
634450!      2!
635520!      1!
636330!      2!
637520!      3!
640310!      1!
641330!      2!
700030!      3!
700050!      1!
700250!      1!
701100!     57!*****
701520!    120!************
702010!      1!
702200!      6!
702730!    149!**************
703020!     13!*
704350!    865!**************************************************************************
704440!     27!**
704560!      1!
705200!      4!
705420!      8!
706110!     78!*******
707110!      1!
710430!      1!
710500!      5!
711240!      2!
712270!      1!
713330!      6!
713340!      2!
714210!      1!
715060!      1!
716700!     24!**
717340!      3!
717350!     12!*
717700!      3!
721170!     13!*
-----------------------------------------------------------------
[DONE]


Programme numero 5 (FORTRAN)

[MYLOOK Monitoring LINK]
command string:...$dd:gau80.cmd/go
how many observations:...40
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  2501
-----------------------------------------------------------------
TOTAL:    4367
RUN:      4017
SLEEP:     350
-----------------------------------------------------------------
```

```
625640!     42!****
625770!     60!******
634450!      4!
635520!      4!
636330!      3!
637520!      1!
640310!      3!
641330!      2!
643240!      3!
700030!      5!
700230!      1!
700250!      1!
701100!    524!**************************************************
701520!    326!********************************
701550!      1!
702730!    434!*******************************************
703020!      8!
703130!      1!
703260!      2!
704320!      3!
704350!   1791!         x100!*****************
704440!     22!**
704620!      1!
704640!      1!
705420!     18!*
706040!      1!
706110!     25!**
707100!      1!
710430!      7!
711240!      5!
711250!      1!
712470!      1!
713330!     15!*
713340!      2!
714210!      1!
715060!      5!
716700!      7!
717340!     23!**
717700!     13!*
-------------------------------------------------------------
[DONE]

Programme numero 13 (COBOL-74)

[MYLOOK Monitoring LINK]
command string:...modirp/go
how many observations:...50
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:   633
-------------------------------------------------------------
TOTAL:     793
RUN:       756
SLEEP:      37
-------------------------------------------------------------
     20!      7!
540000!      1!
540010!     20!**
540030!      3!
541620!      1!
542660!      3!
546330!      6!
547010!      3!
```

```
547100!      3!
553760!      3!
554040!      1!
554640!      4!
554740!      4!
555460!      5!
556130!      1!
557470!      1!
560550!      4!
561300!      2!
562430!      1!
563100!      4!
563130!      1!
564430!      2!
564760!      1!
565510!      1!
567350!      2!
567570!      3!
570220!     14!*
572220!      1!
612470!      2!
621750!     17!*
622110!      9!
623310!     42!****
623700!     10!*
625640!     10!*
625770!     20!**
627210!      7!
635520!      1!
636330!      3!
637520!      2!
640310!      2!
641330!      1!
643240!      3!
644410!      2!
701100!     17!*
701520!     86!********
702730!     93!*********
704350!    250!*************************
706110!     42!****
710500!      2!
710510!      1!
711250!      1!
712230!      3!
716700!      6!
717350!     15!*
717720!      2!
721170!      5!
-------------------------------------------------------------
[DONE]
```

Programme numero 9 (COBOL-74)

```
[MYLOOK Monitoring LINK]
command string:...§paie50.cmd/go
how many observations:...40
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  3967
-------------------------------------------------------------
TOTAL:   7509
RUN:     7319
ioWAIT:     6
SLEEP:    184
-------------------------------------------------------------
     0!    35!***
    20!    14!*
   160!     5!
467260!    12!*
540000!     3!
540010!   134!**************
540030!     3!
541620!     3!
542660!     5!
546330!   236!************************
547010!     5!
547100!     5!
550130!     1!
553760!     5!
554010!     1!
554040!     5!
554630!     1!
554640!     8!
554730!     2!
554740!     4!
555460!     5!
555470!    16!*
556030!     3!
556040!     5!
556050!     2!
556060!     1!
556110!     2!
556130!     5!
557470!    16!*
557540!     1!
560250!     5!
560260!     1!
560270!     1!
560310!     2!
560320!     3!
560550!     4!
560720!     3!
561040!     1!
561210!     1!
561300!     2!
561310!     2!
561320!     7!
561330!     1!
561340!     2!
561430!     1!
561610!     1!
562010!     2!
562430!     2!
```

```
562520!    28!**
562620!     1!
563100!     3!
564420!     2!
564530!     3!
565250!     1!
565510!     8!
565520!     6!
565530!     3!
565560!    11!*
565570!    12!*
565600!     2!
565610!     7!
566660!     5!
566720!    33!***
567240!    49!****
567310!   562!**************************************************************
567320!   285!****************************
567400!     1!
567430!     1!
567440!     1!
567460!     1!
567500!     1!
567700!     1!
567710!     1!
570220!    76!*******
572220!     2!
612470!     7!
621750!    94!*********
622110!    47!****
622720!     2!
622730!     3!
623010!     1!
623100!     1!
623120!     1!
623130!     8!
623230!     1!
623700!    48!****
625640!    41!****
625770!   258!*************************
627210!    61!******
634450!     3!
635520!     5!
636330!     8!
637520!     3!
640310!     7!
641330!     6!
643240!     6!
644410!     3!
700040!     1!
700050!     2!
700130!     4!
700210!     5!
700250!     2!
701040!     2!
701100!   475!************************************************
701520!   732!*********************************************************************
701550!    21!**
702150!     5!
702200!     2!
702620!     1!
702660!     3!
702730!  1124!         x100!***********
```

```
703020!    10!*
703140!     1!
703160!     2!
703170!     1!
703430!    13!*
703510!     1!
704350!  1897!         x100!******************
704440!    47!****
704560!     7!
704600!     1!
704620!     2!
704640!     2!
704660!     1!
704700!     8!
705420!    58!*****
706040!    49!****
706110!    89!********
707100!    24!**
710430!    26!**
711240!    35!***
711250!     1!
711350!     1!
712160!     3!
712230!    63!******
712240!     2!
712270!     3!
712470!    14!*
713270!     8!
713330!    22!**
713340!     1!
714050!     9!
714210!     3!
715060!    14!*
716700!   136!**********
716710!     1!
717700!    31!***
720040!     1!
721410!    45!****
-------------------------------------------------------------
[DONE]
```

4.2.2. Sans PA1050.

Programme numero 2 (FORTRAN)

```
[MYLOOK Monitoring LINK]
command string:...ptfixe,ptfsou/go
how many observations:...40
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  1253
-------------------------------------------------------------
TOTAL:    2631
RUN:      2395
ioWAIT:     17
SLEEP:     219
-------------------------------------------------------------
     0!     53!*****
    20!     22!**
    40!     60!******
   140!     18!*
467260!      9!
540000!      9!
540010!    139!*************
540030!      8!
540260!      1!
541620!     11!*
542660!      9!
542670!      2!
546330!     30!***
547010!     12!*
547100!      9!
547110!      3!
547130!      1!
550130!     13!*
552770!     43!****
553120!    253!*************************
553760!     13!*
554040!     14!*
554640!     25!**
555460!     10!*
556130!      7!
556530!     66!******
557470!     13!*
557540!    304!*******************************
560250!      1!
560550!     11!*
561220!      1!
561300!     14!*
561320!      1!
561330!      1!
561520!      1!
562430!     14!*
563100!     16!*
563130!     11!*
564570!      1!
564610!    475!************************************************
564740!     13!*
565510!     11!*
565600!      1!
567350!      7!
567570!      1!
570220!     47!****
```

```
572220!    11!*
612470!    17!*
621750!   185!******************
622110!    43!****
623310!    60!******
623700!    47!****
625640!    41!****
625700!    12!*
625770!   117!***********
634450!     5!
634560!    14!*
635520!    10!*
636330!    14!*
637520!     5!
640310!     8!
641330!     6!
643240!     7!
644410!     9!
-------------------------------------------------------------
[DONE]
```

Programme numero 7 (FORTRAN)

```
[MYLOOK Monitoring LINK]
command string:...§dd:1105.cmd/go
how many observations:...40
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  4458
-------------------------------------------------------------
TOTAL:  10291
RUN:     9777
ioWAIT:     5
SLEEP:    509
-------------------------------------------------------------
     0!    35!***
    20!    17!*
    40!    62!******
   140!    49!****
   150!     4!
   160!     4!
467260!    10!*
540000!    11!*
540010!   199!******************
540030!     7!
540040!     1!
540050!     2!
540320!     1!
541620!     7!
542660!     5!
546330!   196!******************
547010!    13!*
547100!    13!*
547110!    64!******
550130!    11!*
552770!    37!***
553120!  1519!         x100!***************
553760!    11!*
554000!     1!
554010!     1!
554040!    11!*
554630!     6!
554640!   119!***********
```

```
554770!      2!
555110!      2!
555120!      1!
555460!     12!*
555470!      2!
556040!      2!
556050!      1!
556060!      3!
556130!     14!*
556530!     85!********
557470!     30!***
557540!   5101!          x100!**************************************************
560240!      2!
560250!      2!
560260!      1!
560270!      3!
560310!      3!
560320!      7!
560550!     14!*
560560!      2!
560650!      1!
560670!      2!
560700!      1!
560710!      1!
560720!      2!
561210!      3!
561220!      6!
561300!     17!*
561310!      3!
561320!     10!*
561330!      9!
561340!      6!
561450!      2!
561520!      3!
561530!      1!
561640!      1!
562000!      1!
562160!      1!
562250!      1!
562430!     13!*
563100!      9!
563120!      2!
563550!      3!
563610!      4!
563700!      1!
563750!      1!
564610!    641!************************************************************
564740!     15!*
565510!     23!**
565520!      4!
565530!      8!
565550!      1!
565560!     13!*
565570!     10!*
565600!     11!*
565610!      9!
566660!     10!*
566670!      1!
567350!      3!
567400!      1!
567420!      2!
567430!      1!
567440!      1!
```

```
567470!      1!
567570!      6!
567710!      5!
570220!     50!*****
570230!      1!
572220!      4!
612470!     11!*
612540!      2!
613160!      1!
621750!     77!*******
622110!     40!****
623040!      1!
623310!    161!****************
623700!     36!***
625640!     53!*****
625700!     18!*
625770!    208!********************
634450!      5!
634560!     30!***
635520!     17!*
636330!     15!*
637520!      5!
640310!     11!*
641330!      9!
641430!     37!***
641440!     50!*****
641470!      1!
641500!    345!**********************************
643240!      8!
644410!      6!
-------------------------------------------------------------
[DONE]

Programme numero 13 (COBOL-74)

[MYLOOK Monitoring LINK]
command string:...modirp/go
how many observations:...50
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  640
-------------------------------------------------------------
TOTAL:     658
RUN:       604
SLEEP:      54
-------------------------------------------------------------
     20!      2!
     40!     11!*
    140!     16!*
 467260!      2!
 540000!      1!
 540010!     15!*
 540030!      3!
 541620!      3!
 542660!      2!
 546330!      3!
 547010!      2!
 547100!      2!
 552770!     46!****
 553120!     81!********
 553760!      3!
 554040!      1!
 554740!      1!
```

```
555460!      2!
556130!      3!
556530!     44!****
557470!      3!
557540!     79!*******
560550!      3!
561300!      2!
562430!      2!
563100!      1!
563130!      2!
564430!      1!
564570!      1!
564610!    175!*****************
564760!      1!
565510!      1!
567570!      2!
570220!     12!*
572220!      4!
612470!      3!
621750!      9!
622110!      6!
623310!     13!*
623700!      4!
625640!      8!
625770!      9!
627210!      3!
634450!      2!
634560!      1!
635520!      3!
636330!      4!
640310!      3!
641330!      1!
643240!      3!
-------------------------------------------------------------
[DONE]

Programme numero 9 (COBOL-74)

[MYLOOK Monitoring LINK]
command string:...§paie50.cmd/go
how many observations:...40
interval between looks (in ms):...40
[RESULTS]
mean runtime in ms.:  3802
-------------------------------------------------------------
TOTAL:  10125
RUN:     9840
ioWAIT:     7
SLEEP:    278
-------------------------------------------------------------
      0!    48!****
     20!    45!****
     40!    84!********
    140!   111!***********
    150!     1!
    160!     7!
 467260!    16!*
 540000!     1!
 540010!   143!**************
 540030!     2!
 540040!     2!
 540260!     2!
 540320!     1!
```

```
541620!      3!
542660!      2!
546330!    217!**********************
547010!      6!
547100!      2!
547130!     78!*******
550130!      1!
552770!     36!***
553120!   1821!          x100!******************
553760!      3!
554010!      3!
554040!      5!
554640!      1!
554740!      1!
554770!      1!
555460!     14!*
556030!      1!
556040!      9!
556050!      1!
556060!     22!**
556130!      5!
556530!     38!***
557470!      7!
557540!   2536!          x100!*************************
560250!      5!
560310!      2!
560320!     10!*
560550!      3!
560720!      2!
561050!      1!
561220!      1!
561320!      4!
561330!      7!
561340!      1!
562000!      1!
563100!      3!
564530!      2!
564570!      2!
564610!    217!**********************
564760!      2!
565510!      3!
565520!      4!
565530!      5!
565560!      5!
565600!      2!
565610!      4!
566660!      4!
566720!    135!*************
567030!      3!
567240!     72!*******
567310!   2768!          x100!****************************
567320!    513!**************************************************
567360!      1!
567460!      1!
567470!      2!
570220!     76!*******
572220!      3!
612470!      2!
621750!     90!*********
622110!     40!****
622720!      1!
623000!      1!
623130!      7!
```

```
623240!      1!
623700!     45!****
625640!     54!*****
625700!     65!******
625770!    136!**********
627210!     66!******
634450!      1!
634560!     13!*
635520!     12!*
636330!      4!
637520!      3!
640310!      3!
641330!      2!
641430!     28!**
641440!     51!*****
641500!     66!******
643240!      3!
644410!      2!
```

[DONE]

5. AMELIORATIONS.

5.1. La routine T.14.

```
T.14:   setzm   dcbuf+2         ;read next buffer on next ildb
T.14er: skipn   xbuf            ;If we have an index buffer
        jrst    load            ;no, not the first time
        pushj   p,zxbuf         ;get rid of it

ES$LII: .ERR.    library index inconsistent, continuing
        jrst    load            ;and continue

T.14I:  pushj   p,d.inl         ;read first word
        hlrz    t1,w1           ;block type only
        caie    t1,14           ;is it an index
        jrst    t.14er          ;no, error
        jrst    t.14j           ;don't set flag again

;enter here if in library search mode

T.14A:  skipn   xbuf            ;give error if already been here
        trne    f1,r.inc        ;include being processed
        jrst    t.14            ;process as if no index
        movei   t2,dy.get       ;get space in dy area
        hrrzm   t1,xbuf         ;signal space acquired

T.14J:  hrrz    t1,xbuf         ;aux buffer
        hrli    t1,4400         ;make byte pointer
        movem   t1,xbuf+1       ;and save it
        hrl     t1,dcbuf+1      ;input buffer
        movei   t2,^d127(t1)    ;end of block
        blt     t1,(t2)         ;store block

T.14P:  ildb    w3,xbuf+1
        jumpl   w3,t.14d        ;end of block is negative
        hrrz    w2,w3           ;word count only

        push    p,bg.sch        ;remember current status
        setzm   bg.sch          ;don't search universals

T.14C:  movx    w1,pt.sgn!pt.sym;valid symbol bits
        ildb    w2,xbuf+1       ;get next symbol
        pushj   p,r50t6         ;sixbitize it
        pushj   p,trysym
         caia                   ;no in table, keep trying
         soja   w3,t.14e        ;request matches

T.14K:  sojg    w3,t.14c        ;not required, keep trying

        pop     p,bg.sch        ;restore old status
        ildb    w3,xbuf+1       ;get pointer word
        jrst    t.14b           ;get next prog

T.14E:  move    t1,0(p1)        ;undefined, but do we want it?
        txnn    t1,ps.udf       ;not if already partial defs
        txnn    t1,ps.req       ;certainly not if no request
        aoja    w3,t.14k        ;was A=:2##, don't want a again

        pop     p,bg.sch        ;restore old status

        addm    w3,xbuf+1
```

```
        ildb    t1,xbuf+1
        hrrz    w3,lstblk       ;get last block number
        cain    w3,(t1)         ;in this block?
        jrst    thsblk          ;yes

NXTNDX: skipge  dtaflg          ;different      test for dts
        jrst    nxtdta          ;check if next buffer in core
        cain    w3,-1(t1)       ;next block?
        jrst    nxtblk          ;yes, just do input

T.14f:  useti   dc,(t1)         ;set on block
        wait    dc              ;let I/O finish
        movsi   w2,(1b0)        ;clear ring use bit if on
        hrrz    w3,dcbuf        ;
        iorm    w2,dcbuf        ;set unused ring bit
                                ; (help out monitor)
        skipl   (w3)            ;
        jrts    nxtblk          ;all done now
        andcam  w2,(w3)         ;clear use bit
        hrrz    w3,(w3)         ;get next buffer
        jrst    .-4             ;loop

NXTDTA: wait    dc,             ;let I/O run to completion
        hrrz    w3,dcbuf        ;get pointer to current buffer
        caie    w3,(t1)         ;is it block    we want
        jrst    t.14f           ;no

NXTBLK: IN   DC,
         jrst   thsblk          ;it is now
        jrst    d.err           ;eof or error

T.14D:  hrre    t1,w3           ;get block # of index
        jumpl   t1,eof##        ;finished if -1
        move    t1,w3           ;-1,,block # into t1 for thsblk
        hrrz    w3,lstblk       ;get last block
        jrts    nxtndx          ;check if next buffer in core
```

5.2. Les exemples.

5.2.1. BFRDIO.

Ce programme provient, en partie, du manuel des appels-systèmes TOPS-10 ([MC10]).

```
title bfrdio *** buffered io under TOPS-10 ***
search monsym,iomac
sall


;define 'UUO's

opdef reset [47000000000]
opdef open  [50000000000]
opdef lookup[76000000000]
opdef exit  [47000000012]
opdef releas[71000000000]
opdef in    [56000000000]
opdef getsts[62000000000]

;ac's

tl=1
c=10
j=15

ichn=1

;macro

define  error   (msg)   <
        jrst    [outstr (<
?'msg>)
        jrst    monrtl

>
;param
io.eof=20000

;here on entry - must initialize program

bfrdio: jfcl
        reset

;get time
        movei   1,400000
        runtm
        moveml,runtim
;
        open    ichn,indev      ;associate device with channel 1
         error  (<Can't OPEN device>)
        lookup  ichn,infil      ;find input file to read
         error  (<Can't LOOKUP input file>)

;At this point we could optionally perform INBUF monitor call
;to set up the buffer ring, but instead we'll let the monitor
;do it for us on the first IN monitor call

;here for the main program I/O loop to read the file

io:     jsp     j,gethyt        ;read an input byte
```

```
        jrst    eof             ;end of file reached on input
        jrst    io

;here when input end-of-file reached

eof:    releas  ichn            ;let go of input file and device

;here to return to monitor mode

monrt:  movei   1,400000
        runtm
        sub     1,runtim
        move    2,1
        outl0
;
        exit    1,
        jrst    bfrdio

;Basic low-level buffered I/O routines
;
;       GETBYT is self-initializing e.g. when first called, the
;input byte count is 0 (set by the monitor on the OPEN)
;Getbyt will fall into the IN call to set the default number of
;input buffers and start the device filling them (control will
;return after first buffer is filled).

;GETBYT - routine to read 36-bit input data

getbyt: sosge   incnt           ;any bytes in input buffer?
         jrst   getbuf          ;no - must first read next buffer
        ildb    c,inptr         ;yes - return word in ac c
        jrst    1(j)            ;successful return is skip

getbuf: in      ichn,           ;advance to next buffer
        jrst    getbyt          ;and continue up above
        getsts  ichn,t1         ;oops - got an error
        trne    t1,io.eof       ;was it eof?
         jrst   (j)             ;yes - then not really error
        error   (<input I-O error>)

;all data blocks are defined here
; first: input open block
indev:  exp     13      ;mode=image binary
        sixbit/sys/     ;input device name
        0,,inbdr        ;0,,address of input buffer
                        ; control block
;the LOOKUP block

infil:  sixbit/forlib/  ;input file name
        sixbit/rel/     ;input extension
        block   1       ;prot, mode, creation
        block   1       ;input file ppn

;the I/O buffer control block (the buffers themselves
;will be built by the monitor at run time on execution
;of the first IN UUO)

inbdr:  block   1       ;input buffer control block
inptr:  block   1       ;input buffer ring byte pointer
incnt:  block   1       ;input buffer ring byte count

runtim: block   1
        end bfrdio
```

5.2.2. PMAPIO.

```
title pmapio *** buffered io under tops-20 ***
search iomac,monsym
sall
extern fatal,warn
.require iomac


bytcnt=5
pgscnt=6
bfrcnt=7
from=10


bfrlen=12000
pages=12



pmapio: reset
        move    17,[iowd  ^d20,stack]
;
;get file name
;
        instr   (^d20,14,filspc)
;
;first get the time
;
        movei   1,.fhslf
        runtm
        movem   1,runtim
;
;get jfn of input file
;
        move    1,[gj%old+gj%sht]
        hrroi   2,filspc
        gtjfn
         erjmp  fatal
        movem   1,injfn
        hrlzm   1,from
;
;open the file
;
        move    2,[^d36b5+of%rd]
        openf
         erjmp  fatal
;
;get the file's length
;
        move    1,injfn
        sizef
         erjmp  fatal
        movem   2,bytcnt
        movem   3,pgscnt
;
;map some pages into our buffer
;
map:    move    1,from
        move    2,[.fhslf,,bffpg]
        move    3,[pm%cnt+pm%rd+pm%pld+pages]
        pmap
         erjmp  fatal
```

```
;
;read loop

        setz    bfrcnt,
mnloop: sojl    bytcnt,eof
        cail    bfrcnt,bfrlen
         jrst   eob
        move    1,buffer(bfrcnt)
        aoja    bfrcnt,mnloop
;
;here on End Of File
;
eof:    seto    1,
        move    2,[.fbslf,,bffpg]
        move    3,[pm%cnt+pages]
        pmap
         ercal  warn
        move    1,injfn
        closf
         ercal  warn
;
;get run time
;
        movei   1,.fbslf
        runtm
        sub     1,runtim
        move    2,1
        outl0
;
;ok, done
;
        haltf
;
;here when buffer end reached, pmap some more pages.
;
eob:    addi    from,pages
        jrst    map
;
;data
;
alc     (injfn,1)
alc     (runtim,1)
alc     (filspc,5)
alc     (stack,^d20)
        loc     1000
alc     (buffer,bfrlen)
bffpg=buffer--11
        reloc
end     pmapio
```

5.2.3. Les résultats.

TAILLES DES FICHIERS (en pages)	TEMPS CPU CONSOMMES PAR		GAINS (en %)
	BFRDIO	PMAPIO	
10	90 ms	60 ms	33
150	844	491	42
290	1650	910	45
815	4670	2560	45

5.3. Manipulations de FORLIB avec MAKLIB.

```
§maklib
*forini.rel=sys:forlib.rel/extract:(forini,forpse,cfrxit)
*forlib.rel=sys:forlib.rel/delete:(forini,forpse,cfrxit)

%MKLNIO OUTPUT file DSK:FORLIB.REL will not be INDEXed
*^Z
§append (source file) FORLIB.REL.2 (to) FORINI.REL
 FORLIB.REL.2 [OK]
§maklib
*FORINI.REL=FORLIB.REL/index

*^Z
```

5.4. Les nouvelles mesures.

Nous donnons, ici, les profils d'exécution du LINK pour le même programme en entrée mais avec deux versions différentes de FORLIB. Le premier a été réalisé avec la version courante de FORLIB et le second avec la version modifiée décrite dans la Troisième Partie, 8.6.3.

5.4.1. Exécution du LINK avec l'ancienne version de FORLIB.

Programme numero 4 (FORTRAN)

```
[MYLOOK Monitoring LINK]
command string:...nadnpi/go
how many observations:...100
interval between looks:...40
[RESULTS]
mean runtime in ms.:   640
-------------------------------------------------------------
TOTAL:   1392
RUN:     1366
SLEEP:     26
-------------------------------------------------------------
    20!    24!**
    40!     1!
   140!    17!*
540010!     6!
540030!     1!
540040!     1!
541620!     1!
542660!     1!
546330!     1!
547110!     1!
550130!     1!
552770!   157!***************
553120!   195!*******************
553760!     1!
555460!     1!
556130!     1!
556530!   115!***********
557470!     2!
557540!   183!******************
560550!     1!
561300!     1!
561530!     2!
563130!     1!
564510!     1!
564610!   612!************************************************************
565510!     1!
567350!     1!
570220!     1!
572220!     1!
612470!     1!
621750!     1!
622110!     1!
623310!     2!
623700!     1!
625640!     1!
625770!    17!*
634450!     1!
```

```
634560!      1!
635520!      2!
636330!      1!
637520!      1!
640310!      1!
641330!      1!
643240!      1!
644560!      1!
-------------------------------------------------------------
[DONE]
```

5.4.2. Exécution du LINK avec la nouvelle version de FORLIB.

Programe numero 4 (fortran)

```
[MYLOOK Monitoring LINK]
command string:...nadnpi/go
how many observations:...100
interval between looks:...40
[RESULTS]
mean runtime in ms.:    520
-------------------------------------------------------------
TOTAL:     526
RUN:       519
SLEEP:       7
-------------------------------------------------------------
    20!      2!
   140!      3!
540010!      3!
540030!      1!
540130!      1!
541620!      1!
542660!      2!
546330!      2!
547110!      3!
550130!      1!
552770!    115!***********
553120!    113!***********
553760!      1!
554640!      1!
555460!      1!
556130!      1!
556530!    138!*************
557470!      1!
557540!    106!**********
560550!      1!
561300!      1!
563130!      1!
564510!      1!
565510!      1!
567350!      1!
570220!      1!
572220!      1!
612470!      1!
621750!      2!
622110!      1!
623310!      1!
625640!      1!
634450!      1!
635260!      1!
641330!      1!
```

```
641650!      1!
643240!      1!
644560!      1!
-------------------------------------------------------------
[DONE]
```